Aalto University
School of Science
Master's Programme in Computer, Communication and Information Sciences

Toni Karttunen

# Utilization of prototyping methods in user-centered design process

*Using prototypes to support fast-paced product development in an agile software development project*

Master's Thesis submitted in partial fulfillment of the requirements for the degree of Master of Science in Technology.

Espoo, November 26, 2017

Supervisor: Professor Marko Nieminen, Aalto University
Advisor: Kalle Ikkelä, M. Sc. (Tech.), MBA

Aalto University
School of Science
Master's Programme in Computer, Communication and Information Sciences

ABSTRACT OF MASTER'S THESIS

**Author:** Toni Karttunen
**Title:** Utilization of prototyping methods in user-centered design process

**Date:** November 26, 2017 **Pages:** vii + 87

**Major:** Software and Service Engineering **Code:** SCI3043

**Supervisor:** Professor Marko Nieminen
**Advisor:** Kalle Ikkelä, M. Sc. (Tech.), MBA

To stay competitive in the rapidly evolving business environment, organizations need to be able to create innovations. Novel new products are often created with experimentation, which means that organizations need to use practices that support experimentation. Prototyping is one such practice. Agile software development embraces changing requirements, which makes it suitable for experimentation-driven product development. The overall research problem considers how different types of prototyping approaches can support fast-paced product development in an agile software development project. Research questions include:

1. How to improve prototyping for fast-paced agile software development?
2. How can prototyping support agile requirements engineering?

The research consists of two main parts: literature review and empirical research, which includes action research and interviews.

Prototyping could be improved for the purposes of fast-paced agile software projects by using simplified prototypes and small focused prototypes to make it possible to iterate the design of user interface elements faster. Additionally, low-fidelity prototyping and participatory design could be useful for agile projects. To make large high-fidelity prototypes faster to iterate, better tooling is needed. Prototyping can support agile requirements engineering e.g. by acting as documentation, facilitating communication and by making big picture clearer.

**Keywords:** prototype, prototyping, agile requirements engineering

**Language:** English

Aalto-yliopisto
Perustieteiden korkeakoulu
Master's Programme in Computer, Communication and Information Sciences

DIPLOMITYÖN TIIVISTELMÄ

**Tekijä:** Toni Karttunen
**Työn nimi:** Prototypointimenetelmien käyttö käyttäjäkeskeisessä suunnitteluprosessissa

**Päiväys:** 26. marraskuuta 2017 **Sivumäärä:** vii + 87

**Pääaine:** Software and Service Engineering **Koodi:** SCI3043

**Työn valvoja:** professori Marko Nieminen
**Työn ohjaaja**: Kalle Ikkelä, diplomi-insinööri, MBA

Pysyäkseen kilpailukykyisinä nopeasti kehittyvässä liiketoimintaympäristössä organisaatioiden pitää kyetä luomaan innovaatioita. Uudenlaiset tuotteet saadaan usein aikaiseksi kokeilujen avulla, mistä johtuen on käytettävä käytäntöjä, jotka tukevat kokeilujen tekemistä. Prototypointi on yksi tällainen käytäntö. Ketterä ohjelmistokehitys ottaa halukkaasti vastaan muuttuvat vaatimusmääritykset, joten se soveltuu kokeiluita hyödyntävään tuotteiden kehitykseen. Tutkimusongelma tarkastelee, kuinka erilaiset prototypointitavat tukevat nopeatempoista tuotekehitystä ketterässä ohjelmistokehitysprojektissa. Tutkimuskysymykset ovat:

1. Kuinka prototypointia voidaan kehittää nopeatempoista ketterää ohjelmistokehitystä varten?
2. Kuinka prototypointi tukee ketterää vaatimusmäärittelyä?

Tutkimus sisältää kaksi pääosaa: kirjallisuuskatsauksen ja kokeellisen osan, joka koostuu haastatteluista ja toimintatutkimuksesta.

Prototypointia voidaan kehittää nopeatempoisten ketterien ohjelmistoprojektien tarpeisiin käyttämällä yksinkertaistettuja prototyyppejä sekä pienempiä ja fokusoituneempia prototyyppejä käyttöliittymäelementtien designin iteroinnin nopeuttamiseksi. Matalan tarkkuuden prototyypit ja osallistava suunnittelu voivat myös olla avuksi ketterissä projekteissa. Isojen korkean tarkkuuden prototyyppien iteroinnin nopeuttaminen vaatii uusien työkalujen kehittämistä. Prototypointi voi tukea ketterää vaatimusmäärittelyä esim. toimimalla dokumentaationa, helpottamalla kommunikaatiota ja tekemällä ns. ison kuvan selvemmäksi.

**Avainsanat:** prototyyppi, prototypointi, ketterä vaatimusmäärittely

**Kieli:** englanti

# Acknowledgements

Creating this thesis was a long journey, which included some challenges. Writing the thesis required significant effort, which would not have been possible without support from my thesis supervisor professor Marko Nieminen and advisor Kalle Ikkelä. Additionally, I would like to thank my workplace RND Works for their support for my thesis and our client company Veikkaus that provided environment for conducting the empirical part of the thesis.

While writing this paper, I am finishing my Master's degree studies. When I started my studies in Otaniemi, I could not have imagined how many memorable experiences I would gain during my studies. I want to thank all the fellow students and university employees that I met during that time.

Additionally, I would like to thank my family and friends who have supported me throughout my studies.

Espoo, November 2017

Toni Karttunen

# Contents

# Chapter 1 Introduction

## 1.1 Background

Modern software development organizations face rapidly evolving competition. Additionally, fast changes in technology landscape and time pressure cause challenges, which today's product development teams have to face. Due to the constantly changing requirements, traditional requirements engineering approach that tries to discover all the requirements upfront is not feasible in many software projects nowadays. (Boehm, 2000)

To succeed in the harsh competition, product development teams must be able to elicit, analyse, validate and refine requirements quickly. Agile requirements engineering faces these challenges with a set of practices that attempt to solve the challenges of the fast-paced business and technology environment; these practices include prototyping, iterative requirements engineering, face-to-face communication, extreme prioritization, constant planning and reviews and tests (Ramesh, Cao, & Baskerville, 2010).

Experiment-driven problem solving can give competitive advantage to organizations that attempt to be create innovations. A typical approach to problem solving when creating novel products is to create a simplified model of a product (e.g. in form of a prototype) and find out through trial and error if it works. If it is noticed that the design does not work, the model is refined iteratively until it works. The benefit of the approach of using simplified models instead of the design of the whole product is that a simplified model does not need to include those parts of the product that are not relevant in terms of the problem to be solved. (Thomke, von Hippel, & Franke, 1998) Coming up with successful novel ideas is difficult by just analysing and studying a phenomenon; experimentation-driven approach is more likely to yield good results (Ries, 2011).

Applicability of experimentation-driven problem solving is not limited to organizations that attempt to be innovative. For example, prototyping can act as a tool for managing risk (Schrage, 1993), which can be useful for risk-averse organizations. Prototypes are usually faster to build than a real product, which makes it possible to evaluate the product with relevant stakeholders before committing resources to building the real product.

Prototyping, an agile requirements engineering practice, can be useful for various reasons: for example, it enables early validation of designs (Drews, 2009) and early involvement of users and customers in product development process (Brown, 2008). Prototyping facilitates communication between product development team and customer (Käpyaho & Kauppinen, 2015). Moreover, many agile projects have replaced written documentation with prototypes (Ramesh et al., 2010).

## 1.2 Research problem and research questions

### 1.2.1 Research problem

As it was stated previously in section 1.1, creating innovative new products at a rapid pace is important for organizations that want to remain competitive. Organizations need to be able to iterate their product designs fast in order to respond to rapid changes of business needs and technology landscape. Many organizations have moved to agile approaches in their requirements engineering process to support creation of digital products in rapidly changing business and technology environments. The overall research problem considers **how different types of prototyping approaches can support fast-paced product development in an agile software development project.**

Fast-paced agile landscape causes some unique challenges to prototyping, such as need to be able to modify a prototype frequently due to frequent requirement changes. Due to the popularity of agile methodology in software development projects nowadays, it is necessary to discover how prototyping should be practiced in agile projects in order to be efficient. It is also useful to consider how prototyping can support agile requirements engineering.

Prototyping can be done with various approaches and the choice of approach can influence e.g. the speed of creating and modifying the prototypes (Houde & Hill, 1997). Therefore, it is important to consider, which approach is used so that the goals of fast-paced software development projects can be met and prototyping serves the purposes of the project instead of acting as a bottleneck.

### 1.2.2 Research questions

Prototyping is approached from multiple angles in this research. The main research questions include:

**RQ1: How to improve prototyping for fast-paced agile software development?**

Prototyping can be challenging in agile software projects. Constantly changing requirements require ability to create and refine prototypes quickly. Minimal documentation can make it difficult to understand what the prototype should

even prototype if customer is not able to participate in the project actively to clarify the requirements. Focus on short-term time-to-market planning can make it difficult to understand what features will be needed in the future (Ramesh et al., 2010)

Conventional knowledge of prototyping implies that low-fidelity prototypes can be created faster than high-fidelity prototypes, which makes them more suitable for situations when it is necessary to explore a large number of alternative designs or when it is desirable to be able to create prototypes at a rapid pace (Snyder, 2003). During the recent years, a plethora of new high-fidelity prototyping tools have entered the market, which begs us to question if high-fidelity prototyping has become viable for fast-paced agile projects.

Prototype fidelity is not the only factor that affects suitability of prototyping for fast-paced agile software development; e.g. the focus of the prototype may have an effect on how the prototype is perceived by customer and end users and how fast the prototype can be created (Houde & Hill, 1997). Overall, finding the adequate balance of different factors is important when utilizing prototyping in agile projects.

### RQ2: How can prototyping support agile requirements engineering?

When utilizing agile requirements engineering, multiple challenges may occur: for example, a customer may not trust the product development team due to not having comprehensive documentation and verifying the requirements can be difficult when comprehensive documentation does not exist. Moreover, because agile requirements engineering relies heavily on having good communication, challenges in communication can have a negative impact on an agile project. Additionally, understanding non-functional requirements can be challenging and understanding the overall high-level goals of the project can be difficult. (Ramesh et al., 2010)

Due to a plethora of challenges in agile requirements engineering, it is useful to ask if agile requirements engineering practices, such as prototyping, could be used to combat the previously mentioned challenges. Some of these ways how prototyping can help with agile requirements engineering, such as acting as documentation in place of written requirements documents are quite obvious, but more research is needed to have a better understanding of how

prototyping can help with the agile requirements engineering challenges and if the chosen prototyping approach has any effect on how prototyping can help.

## 1.3 Scope

The scope of the thesis is limited to design of graphical user interfaces in software products. The empirical study focuses on using the prototyping methods for designing a mobile application user interface, although it is assumed that the majority of the results that are gathered in the case study are applicable to graphical user interfaces on at least some other platforms, such as conventional web user interfaces. When discussing prototyping, the main focus is on user interface prototyping, but some other forms of prototypes, such as technical proof-of-concept prototypes are mentioned in those situations where is considered useful to compare user interface prototypes with other types of prototypes.

The empirical part of this thesis was done at one agile project at a Finnish company, in which I was designing a mobile gaming application. Because the empirical research only covers one project at one company, there are some limitations related to the generalizability of the findings.

Some types of experimental graphical user interfaces, such as virtual reality and augmented reality user interfaces, are left out of this study because they will probably require using different prototyping tools instead of those tools that are popular for designing user interface prototypes for conventional mobile applications. Moreover, it was not possible to prototype such user interfaces in the empirical study project due to the constraints of the project environment.

# Chapter 2 Literature review

This chapter presents an overview of the existing scientific knowledge of prototyping and agile requirements engineering. To make it easier to reproduce the results of the literature review, section 2.1 describes how data was collected and how the literature was selected for the literature review.

Because prototypes can be defined in multiple different ways according to different sources, a definition of the topic is provided in section 2.2 to make it clear how prototypes are defined in this study so that it is possible to understand that if some other study has defined prototypes in a different way, the results of the study may be different due to a different definition. To understand why prototyping is done, section 2.3 describes the benefits of prototyping. To provide a balanced view of the topic, the drawbacks of prototyping are described in the following section 2.4. Prototyping can be done with different levels of fidelity and the chosen level of fidelity can impact how fast prototypes can be created and modified and therefore prototype fidelity can affect how well prototyping works in a fast-paced agile software development project, in which requirements change frequently. Prototype fidelities are presented in section 2.5.

To provide a brief introduction to the agile way of thinking to those readers who are unfamiliar with the topic, a short description of agile methodology is available in section 2.6. Agile requirements engineering is used to face the problems that are caused by the challenges of many today's software projects: rapid changes in technology and customer's and user's needs and limited amount of time (Cockburn, 2002; Ramesh et al., 2010). The practices that agile requirements engineering utilizes to face these challenges are described in section 2.7.1. Unfortunately, applying the agile requirements engineering practices also cause some problems; these problems are presented in section 2.7.2.

To summarize the answers to the research questions based on the literature, the chapter ends with description of how to improve prototyping for agile software development and how prototyping can support agile requirements engineering.

## 2.1 Methodology for literature review

To improve reproducibility of literature review, a systematic approach of identifying and analysing relevant literature was needed. Staples & Niazi describe how to conduct a literature review by using systematic review guidelines similarly as it is often done in other fields of science (e.g. medicine), in which it is systematic literature review guidelines are more established than in software engineering research (Staples & Niazi, 2007). The review process that Staples & Niazi described was modified for this thesis to better fit the needs and requirements of a typical Master's thesis. The modified version of the systematic literature review process that I used is described in sections 2.1.1–2.1.2.

### 2.1.1 Data collection

The literature search consisted of two phases. The first phase, initial literature search, was done by using a large number of potential search terms ("keywords") to understand which search terms could return enough relevant results so that they could be used for this study. The second phase, final literature search was done by using the search terms that are listed in table 2.1. The literature search was conducted by querying Aalto Learning Centre databases, Elsevier Mendeley literature search tool and Google Scholar.

When relevant primary articles were found, additional literature was searched by using the references of the primary articles and by searching the other publications of the author of the primary articles (i.e. by using the commonly known "snowballing" technique for finding literature). Additionally, a small amount of literature was added by using the course material literature of Aalto University usability courses and service design courses.

*Table 2.1:* Search terms that were used for searching literature.

| **Search term** |
| --- |
| prototyping |
| prototype |
| prototyping AND “user interface” |
| prototype AND “user interface” |
| software prototyping |
| prototyping AND fidelity |
| prototype AND fidelity |
| “paper prototyping” |
| “paper prototype” |
| “low-fidelity prototyping” |
| “low-fidelity prototype” |
| “high-fidelity prototyping” |
| “high-fidelity prototype” |
| “mixed-fidelity prototyping” |
| “mixed-fidelity prototype” |
| “multi-fidelity prototyping” |
| “multi-fidelity prototype” |
| prototyping AND mobile |
| prototype AND mobile |
| agile AND UX |
| agile AND “user experience design” |
| “requirements engineering” |
| agile AND “requirements engineering” |
| “agile requirements engineering” |
| “agile requirements engineering” AND challenges |
| prototyping AND “requirements engineering” |
| prototyping AND agile AND “requirements engineering” |
| “agile software development” |

### 2.1.2 Literature selection criteria

After search results were received from a search query, the first step was to discard clearly irrelevant articles, in which even the title did not include relevant topics. Then, the abstracts of the articles were read to evaluate whether the article at hand was potentially relevant. Finally, the contents of the articles were evaluated to decide if the articles were relevant for this study.

When reviewing the results of the search queries, more value was given to articles that high citation counts (unfortunately, this information was only available in certain search tools and therefore the citation counts were not examined for all the literature). If two similar articles were found and one of them was peer-reviewed and the other was not, the peer-reviewed article was considered more trustworthy and it was chosen based on that evaluation criteria.

## 2.2 Definition of prototypes and prototyping

Before starting to explore prototypes and prototyping in more detail, it is useful to define what they mean because different sources have somewhat different definitions for the topic; some definitions are quite narrow and strict, whereas other definitions are broader and more ambiguous. Therefore, it is important to have a clear definition of the topic to avoid misunderstandings when comparing the results of this thesis with the results of other research about the topic.

Handbook of human-computer interaction (Houde & Hill, 1997) has a very broad definition of a prototype: according to the handbook, a representation of a design idea is a prototype, regardless of the implementation medium of the prototype. Therefore, a prototype can be defined in multiple ways and it can mean different things to different groups of people; for instance, an interaction designer may call "a simulation of onscreen appearance and behaviour" a prototype, whereas a user researcher may call a storyboard a prototype.

In Snyder's book Paper Prototyping (2003), a prototype is defined more narrowly; according to Snyder, e.g. a storyboard is not a prototype because it is not possible for a user to interact with a storyboard; according to Snyder, [paper] prototyping is a form of usability testing. Houde & Hill (1997) state that a storyboard can be a prototype because it shows a design idea in a concrete way and facilitates discussion about design problems. Due to the fact that this thesis discusses the applicability of using prototypes for requirements engineering, not just for usability testing, this thesis follows Houde & Hill's (1997) definition of the topic. Prototyping is defined as the activity of creating prototypes in this thesis.

## 2.3 Motivation: benefits of prototyping

The goal of making prototypes is to get answers to design questions. Prototypes enable exploring various design alternatives. Prototypes help facilitating design discussions around a concrete artefact. (Houde & Hill, 1997; Schrage, 1993) Prototypes can also act as a means for exploring new design questions (Schrage, 1993).

Because it is usually faster to make a prototype than the real product, prototyping supports early user involvement in product development process (Brown, 2008). Early validation of designs can help manage risks by detecting problems before making commitment to a flawed design (Drews, 2009; Schrage, 1993). Prototypes help organizations test more ideas than they could by implementing the real product without doing a prototype first (Schrage, 1993). Testing multiple design alternatives is useful, because when users can view multiple alternatives, they give more realistic (i.e. more critical) feedback in usability tests than when viewing just one design alternative (Tohidi, Buxton, Baecker, & Sellen, 2006).

Prototypes are essentially simplified models of a real product that do not need to contain all those details of the real product that are expensive to model and potentially not relevant to the design task at hand. Due this simplification—leaving out irrelevant information—prototypes can be easier to analyse than the real product. (Thomke et al., 1998)

In agile software projects, it can be challenging to manage requirements with very little written documentation. Prototypes can help with this challenge by acting as a form of documentation and by acting as a tangible artefact to facilitate discussion about requirements. When used as documentation, prototypes can reduce ambiguity of requirements and create trust between the customer and product development team. (Käpyaho & Kauppinen, 2015)

Prototypes can help having quality communication with customer, which is important in agile projects that rely heavily on having good communication. (Käpyaho & Kauppinen, 2015) Using prototypes in participatory design can help involving a multidisciplinary team in the design process (Snyder, 2003).

Additionally, updating prototypes can feel more motivating than updating written documentation. Moreover, if prototypes are implemented with the same technologies as the final product, prototypes can save time when implementing the final product. (Käpyaho & Kauppinen, 2015)

## 2.4 Problems and challenges

Houde & Hill (1997) argue that due to the complexity of creating prototypes of interactive computer systems, it can be challenging or even impossible to prototype a whole design. Therefore, successful prototyping often requires finding suitable focus for the prototype and communicating the focus and

limitations of the prototype to various stakeholders, such as other members of a design team, users, product managers and business stakeholders. Unfortunately, it can be difficult to explain which parts of the system correspond to the actual artefact. There is a chance that the effects of the other interrelated parts of the product that affect user experience of the whole product are not understood when evaluating a prototype that focuses on certain aspects of the product. Moreover, as Houde & Hill (1997) explain, prototypes are not always self-explanatory, which may cause challenges if a prototype is presented without sufficient explanation about its purpose and what it is supposed to prototype. Additionally, it is possible that a prototype does not correspond the actual artefact realistically enough from user's/customer's point of view, which may lead to misunderstandings.

Additionally, Houde & Hill (1997) state that focusing on only a certain part of the system may not yield correct results when evaluating the user experience of the product, because interactive computer systems consist of a large number of interrelated parts, which affect the user experience of using the product.

In some organizations, the organization's culture may only view certain types of prototypes valid or prototypes' purpose is to make a point instead of acting as a mean for facilitating discussion. (Schrage, 1993) In such organizations, all the benefits of prototyping cannot be accomplished.

If prototyping media (or prototyping tools) are chosen so that the prototypes are difficult and slow to modify (prototypes become *"untouchable works of art"*), prototyping does not encourage exploration of new ideas so well as prototyping media that enables easy and rapid modification of the prototypes. The quantity of prototypes that an organization is able to produce and the speed of how fast an organization can produce a prototype tell about how fast the organization is able to explore new ideas. If creating a prototype takes very long time, it is likely to be treated as an end result of thinking process instead as a means for exploring opportunities. (Schrage, 1993)

Putting lots of focus on visual user interface prototyping can be harmful for system architecture and lead to putting not enough focus on non-functional requirements and quality issues. Moreover, the speed of creating prototypes may cause non-technical customers to have too high expectations for development speed of the real product. (Käpyaho & Kauppinen, 2015)

If prototype is implemented in code, there is a temptation to use prototype code in a real product to speed up the development process. Because prototypes are usually not implemented as robustly as real products, prototype

code may cause risks when used in production because prototypes are likely not as secure, scalable and robust as real products. (Ramesh et al., 2010)

## 2.5 Prototype fidelity

As described by Houde & Hill (1997), prototype fidelity means how finished the behaviour and the visual look of the prototype are i.e. how close the design of the prototype is to the eventual design of the actual product. According to their definition, prototype fidelity does not mean the amount of detail; the term that is used to describe the amount of detail is *resolution*.

It is important to note that prototype fidelity is not a measure of the completeness and readiness of a design. As an example, Houde & Hill (1997) mention that a high-fidelity prototype may be used for market research during the early stages of the design process, whereas a low-fidelity prototype may be used to evaluate the structure of the user interface in the later stages of the process. Additionally, it is worth noting that the level of fidelity may not correspond to the complexity of the open design question that the prototype is supposed to answer; sophisticated questions may be answered with the help of a simple prototype (Schrage, 1993).

### 2.5.1 Low-fidelity prototyping

Paper prototyping, perhaps the most well-known form of low-fidelity prototyping, is a platform-agnostic method for designing, creating and testing user interfaces. It can be also useful for ideation (co-design and brainstorming) and communication with various stakeholders. (Snyder, 2003)

As defined by Snyder (2003), paper prototyping means usability testing, in which the user performs realistic tasks with a paper-based version of the user interface. In this form of usability testing, a human “acts as a computer” by simulating the tasks that the user performs with the paper-based interface. In a test session, there is typically a facilitator that conducts the test session and other team members observe the session and take notes. In the test session, it is not supposed to explain to the user how the user interface is supposed to work—rather, the purpose is to simulate the computer’s actions based on user’s input and observe how the user interacts with the user interface. Even though paper prototypes are usually created for the purposes of usability testing,

Snyder (2003) mentions that paper prototypes can also be useful for other purposes, e.g. internal reviews.

A paper prototype can consist of screenshots or hand-drawn sketches of the user interface. The presentation medium of the prototype can be e.g. paper or a whiteboard. According to Snyder's definition, storyboards, wireframes, and compositions ("comps") are not paper prototypes, even though it is possible to turn them into paper prototypes. Compositions are usually used for product design organization's internal communication about visual design and they often contain placeholder content that makes them unsuitable for usability testing. Similarly, wireframes usually only contain placeholder content, because their purpose is to describe page layout and navigation. Storyboards are essentially flowcharts, which describe the flow of user's tasks and how the user interface supports accomplishing the tasks. Because it is not possible for users to interact with storyboards, storyboards are not paper prototypes. (Snyder, 2003)

According to Snyder (2003), paper prototyping has several benefits. Because it is fast and easy to create paper prototypes, paper prototyping supports rapid iterative development and examination of multiple alternative versions of a user interface. Additionally, the speed of paper prototyping makes it possible to receive feedback from users during the early in the development process without making investment into any specific implementation.

Due to its simple and easy-to-use nature, paper prototyping can be useful for multidisciplinary product development teams because it enables people to participate in the design process regardless of their educational and professional backgrounds. Moreover, paper prototyping can support communication within the team and between the team and stakeholders. (Snyder, 2003)

Low-fidelity prototyping has various benefits. According to Houde & Hill (1997), low-fidelity prototyping makes it possible to focus on high-level goals, such as overall functionality of a product instead of paying lots of attention to implementation or look and feel.

Unfortunately, low-fidelity prototyping has some drawbacks, too. Because low-fidelity prototypes are not always self-explanatory, some stakeholders may misunderstand the purpose and focus of the prototype and they may spend their attention on commenting on irrelevant details of the prototype that are not even supposed to correspond the eventual artefact that is being prototyped. (Houde & Hill, 1997)

### 2.5.2 High-fidelity prototyping

There are some advantages in using high-fidelity prototypes when compared with low-fidelity prototypes. Houde & Hill (1997) argue that because the visual design of high-fidelity prototypes is closer to the real artefact that is being designed, they are suitable for gathering feedback from people with various backgrounds. They claim that high-fidelity prototypes can be useful when presenting the prototype to people who are unfamiliar with prototypes. Additionally, because high-fidelity prototypes correspond to the real artefact that is being designed better, the likelihood that people misunderstand the prototype can be lower than with low-fidelity prototypes (when evaluating low-fidelity prototypes, people may not understand what is supposed to be "real" and in the focus of the prototype if the purpose of the prototype is not clearly communicated). If we generalise this notion of high-fidelity prototypes being useful for situations when it is challenging to communicate the purpose of the prototype to the user, we can understand why high-fidelity prototypes can useful for e.g. market research and remote (unmoderated) usability testing.

In addition to the previously mentioned situations, high-fidelity prototypes can be useful for some purposes for which low-fidelity prototypes are completely inapplicable, such as evaluating the look and feel of the artefact before having to make a decision to commit to developing it for real. If a high-fidelity prototype is done as an implementation prototype, it can also help assessing technical feasibility or performance of the product to act as a proof-of-concept type demonstration. (Houde & Hill, 1997)

### 2.5.3 Multi-fidelity prototyping

Creating a clear distinction between fidelity levels is not practical at times. For example, it is possible that some aspects of a prototype need to have very high fidelity (e.g. correct data and logic), whereas it may not be necessary to implement the other aspects of the prototype (e.g. visual look and feel) with high fidelity. In these cases, it can be difficult to categorize a prototype as a high-fidelity prototype or a low-fidelity prototype. (McCurdy, Connors, Pyrzak, Kanefsky, & Vera, 2006)

Sometimes, a mixture of multiple prototype fidelities is the most convenient choice. When involving stakeholders with various backgrounds in the prototyping process, it is likely that the different stakeholders have varying

prototyping skills: some members may only be able to do prototyping with pencil and paper, whereas some other stakeholders may be able to create polished digital prototypes. In these scenarios, it may not be always a good idea to limit the whole team to the lowest common denominator but use an approach that takes into account the prototyping skills of different people better. It is possible to do prototyping with tools that combine sketches and digital drawings to support multiple fidelity levels within a prototype. (Coyette, Kieffer, & Vanderdonckt, 2007)

It is also possible to take the idea of Coyette, Kieffer and Vanderdonckt further by enabling in-situ participatory design with end users by creating a mixed-fidelity prototype that runs on mobile devices and utilizes active data collection (e.g. in the form of experience sampling) or passive data collection (e.g. logging) (de Sá, Carriço, Duarte, & Reis, 2008). Traditional paper prototyping requires having a usability test facilitator, which can limit its applicability to certain situations. "Paper on screen" mixed-fidelity prototyping can be useful in those scenarios because it does not require a facilitator. (Bolchini, Pulido, & Faiola, 2009)

Another approach to multi-fidelity prototyping is creating a partial medium-fidelity wireframe prototype for a usability test and utilizing participatory design with users who are asked to imagine what should be in some parts of the prototype that are intentionally left blank. (Still & Morris, 2010)

## 2.6 Agile methodology

Agile software development emphasizes reacting to changes instead of trying to follow a plan. Additionally, agile software projects aim at delivering working software at frequent intervals instead creating documentation. According to agile principles, achieving better customer satisfaction and co-operation between development team and business stakeholders is more important than spending time for negotiating contracts. (Agile Alliance, 2001; Paetsch, Eberlein, & Maurer, 2003)

## 2.7 Agile requirements engineering

Traditional requirements engineering process consists of a few key activities: requirements elicitation, requirements analysis and negotiation, requirements documentation, requirements validation and requirements management. (Paetsch et al., 2003; Ramesh et al., 2010)

In agile requirements engineering, requirements elicitation happens iteratively, whereas traditional requirements engineering attempts to find all the requirements upfront at the start of the project. In agile requirements engineering, requirements analysis and negotiation emphasizes prioritizing, changing and refining requirements, whereas in traditional requirements engineering the focus of this activity is on solving conflicts in requirements. Traditional requirements engineering attempts to create comprehensive and complete documentation, but in agile requirements engineering documentation is minimal and there is no formal requirements documentation activity. Agile requirements engineering tries to evaluate if the requirements meet user needs in the requirements validation activity, whereas traditional requirements engineering focuses on creating a complete and consistent document in the requirements validation activity. (Ramesh et al., 2010)

### 2.7.1 Agile requirements engineering practices

Time pressure, evolving requirements and rapidly changing technology are common challenges in agile projects (Cockburn, 2002; Ramesh et al., 2010). These challenges are the reason why the following six practices are used in agile requirements engineering (Ramesh et al., 2010):

**Iterative requirements engineering:** Requirements emerge during the whole product development process; there is no intention to capture all the requirements of the product at the beginning of the project. Some high-level analysis of the requirements is carried out during the early phases of the project, but the purpose of this planning is to gather requirements for the first releases of the product instead of documenting all the requirements for the whole duration of the product development project. (Ramesh et al., 2010)

**Constant planning:** Reacting to change in the environment constantly during the product development process is emphasized. Typically, two types of requirement changes can occur: (1) some features are dropped or added and (2) existing features are modified. (Ramesh et al., 2010)

**Prototyping:** Prototypes are used as a means of communication with the customer instead of traditional requirements documents. Requirements validation and refinement are done based on prototypes. (Ramesh et al., 2010)

**Face-to-face communication:** Traditional requirements documents are replaced with face-to-face communication between customer and product development team in agile projects. When written documentation is used, it usually appears in the form of user stories or other simple techniques. An exception is mission-critical systems, such as banking software, in which formal written oral documentation may be needed. (Ramesh et al., 2010)

**Extreme prioritization:** Features with the highest business impact are developed first. Requirements prioritization is done frequently during the project, not just at the beginning of the project like in traditional requirements engineering. (Paetsch et al., 2003; Ramesh et al., 2010)

**Reviews and tests:** In review meetings, project's progress is demonstrated to understand if the project is moving towards the desired goal, to learn about potential problems early enough and to increase customer's trust in the project (Paetsch et al., 2003; Ramesh et al., 2010). Acceptance tests are used as a means of requirements validation and verification. *Acceptance Test-Driven Development (ATDD)* and *Test-Driven Development (TDD)* are commonly used in agile projects. (Ramesh et al., 2010)

### 2.7.2 Agile requirements engineering challenges

Utilization of the previously mentioned six agile requirements engineering practices leads to various challenges, which are described below.

**Minimal documentation** can cause challenges when some team members leave the project or new people join the project. Additionally, it is difficult to do formal verification when traditional requirements documentation does not exist. Moreover, some customers may not trust the process due to only having small amounts of written documentation. (Ramesh et al., 2010)

Constant requirements engineering work can be challenging in terms of **motivation** (Bjarnason, Wnuk, & Regnell, 2011). Additionally, some developers may not be accustomed to applying the testing methods that are commonly used in agile software development (Ramesh et al., 2010).

Having to make changes to inadequate system architecture during the late stages of the project, difficulty to estimate project's schedule and cost and spending vast amounts of effort for refactoring are common problems in agile projects (Ramesh et al., 2010). These issues may be a sign of **missing big picture**.

Because agile projects emphasize **good quality communication** instead of documentation, having sufficient customer access and participation are necessary, but often there are problems with getting customer involved in the project actively enough (Paetsch et al., 2003; Ramesh et al., 2010). Sometimes, getting multiple customer groups to agree on requirements can be challenging, too. (Ramesh et al., 2010)

Because requirements are prioritized based on business value, often with heavy time-to-market focus, it is possible that there is not enough focus on **non-functional requirements** in many agile projects (with exception of usability). (Ramesh et al., 2010)

Prototyping itself can cause some problems. Customers may have too high expectations based on user interface prototypes. Additionally, using prototype implementation of a feature in a real product can cause quality issues. (Ramesh et al., 2010)

## 2.8 How to improve prototyping for fast-paced agile software development?

There are numerous challenges that must be faced when doing prototyping in fast-paced agile software development projects. Naturally, fast-paced projects require that prototyping can be done quickly so that prototyping does not slow down the fast speed of the project. Moreover, because agile methodology emphasizes *"responding to change over following a plan"* (Agile Alliance, 2001), it is essential for prototypes to be able to be modified quickly after frequent requirement changes. Minimal documentation can cause issues whenever there is a problem with communication, such as not having access to right customer representatives or difficulty of knowing requirements when a team member leaves the project or new people join the project. Moreover, focusing on short-term time-to-market planning may cause a need for laborious changes later during the project to modify the product for the new requirements, which had not been anticipated earlier. Time pressure may also cause teams to use prototype implementations of products is real production software (Ramesh et al., 2010)

Luckily, it is possible to improve prototyping for fast-paced agile software projects in various ways. A summary of how to improve prototyping for fast-paced agile projects is available below.

**Choose a focus for your prototype when it makes sense:** Due to the complexity of modern computer systems, creating a prototype that represents a whole system can be laborious and time-consuming. Based on the notion that prototypes are a tool for studying design problems and assessing solutions, Houde & Hill (1997) recommend that the focus of the prototype should be based on what the *most significant open design questions* are. If a prototype focuses on a certain aspect of the product that is being designed, a vital part of successful prototyping is to communicate which parts of the prototype correspond to the artefact and which do not.

According to Houde & Hill (1997), a prototype can focus on either the *role* of the product in users' lives, *implementation* or *look and feel*. Focusing on the role can be useful, if the purpose is to create some new functionality and the open design questions are related to what the role of the product is supposed to be in users' lives and what features could potentially support it. When the task at hand is to present some functionality of an already known role in a new

way, the focus can be on the look and feel of the product. Focusing the implementation can be relevant in those cases when a new technique is used as a basis of the functionality of the artefact.

Sometimes, approaching a design problem from a single point of view does not yield good results. Therefore, Houde & Hill (1997) recommend creating multiple prototypes (potentially with different focus) to approach a design problem from multiple viewpoints.

However, after testing the effect of focused prototypes on a design problem, **it is also important to test how the whole system works together,** because the interactions of the interrelated pieces of a complex system may affect the results. (Thomke et al., 1998)

**Use different levels of prototype fidelity based on the needs:** Low-fidelity paper prototyping is fast and therefore it supports rapid iterative development and exploration of multiple alternative versions of a user interface. Because paper prototypes can be created very quickly, they make it possible to get feedback from users even during the very early phases of development process. (Snyder, 2003)

According to Houde & Hill (1997), low-fidelity prototyping makes it possible to focus on high-level goals, such as overall functionality of a product. Therefore, it can be useful when thinking of the big picture, which can be a challenge in agile projects. Snyder (2003) argues that a simplified model of a user interface can be easier to analyse than a prototype that contains lots of detail. Moreover, simplified prototypes can be suitable for keeping customer's expectations of development speed realistic, which can be a challenge is agile projects (Käpyaho & Kauppinen, 2015).

Requirements change iteratively and frequently in agile projects (Ramesh et al., 2010). Due to the constant change of requirements, it can be difficult to keep a large high-fidelity prototype up-to-date. Thus, low-fidelity prototypes and focused prototypes can be easier to keep up-to-date in agile projects. Snyder (2003) argues that after a high-fidelity prototype has been created, there is a tendency to avoid modifying it because it can take a lot of time to modify a high-fidelity prototype.

However, high-fidelity prototypes also have some advantages when compared with low-fidelity prototypes. Since the visual look and feel of high-fidelity prototypes resembles the design of the real artefact more closely, they are more suitable than low-fidelity prototypes for those occasions when

prototypes are shown to people with versatile backgrounds. (Houde & Hill, 1997)

When choosing the focus and the level of fidelity of the prototype, it is vital to understand to whom the prototype is shown, because the expectations of these people (perhaps based on the culture of the organization) can affect how they view the prototype. To mitigate the problems that may arise when showing low-fidelity prototypes to people, it is a good idea to explain clearly what design problem you are trying to solve and perhaps even more importantly, what aspects you are not trying to solve so that the people to whom the prototype is shown can focus their attention on those issues that you are interested in. (Houde & Hill, 1997) To support rapid exploration of design alternatives (one of the main purposes of prototyping), prototyping media should be chosen so that it is easy and quick to modify the prototypes (Schrage, 1993).

When speed of learning based on experiments is important, it is a good idea to utilize **parallel experimentation** (test multiple prototypes at once) instead of utilizing serial experimentation (test one prototype at a time and only start doing a new prototype after testing the previous one). (Thomke et al., 1998) Parallel experimentation can also produce more critical (i.e. realistic) feedback in usability testing (Tohidi et al., 2006).

**Utilize participatory design:** In agile software development projects, comprehensive requirements documentation does not usually exist and due to the constant changes of requirements it can be difficult to keep track of the requirements (Ramesh et al., 2010). Therefore, it can be challenging for the person who creates the prototype to understand all the requirements thoroughly. Collaborative prototype-driven problem solving can be used to engage various stakeholders in the prototyping process (Bogers & Horst, 2014). Snyder (2003) suggests that paper prototyping is a good approach for multidisciplinary design work because paper prototyping does not require any technical skills from people who want to participate in the prototyping process.

**Treat prototypes as disposable experiments and communicate it clearly:** If prototypes are created as implementation prototypes (with some actual technical implementation), it is advisable to treat them as disposable experiments. Otherwise, the prototype implementation that is included in the

actual product can become challenging to maintain and develop further. Moreover, the user interfaces of implementation prototypes may not be properly designed and if they are not redesigned to make them good enough for actual production usage, the user experience may suffer. (Houde & Hill, 1997) It is a common problem in agile projects that prototype code is used in production software due to time pressure (Ramesh et al., 2010). Because all the non-technical stakeholders do not always understand that prototypes are usually supposed to be disposable experiments, it is necessary to communicate very clearly that a prototype is not a final product.

## 2.9 How can prototyping support agile requirements engineering?

Prototyping can support requirements engineering in several ways. Given that the purpose of creating prototypes is to get answers to design questions and act as a communication facilitator between designers, developers and other stakeholders (Houde & Hill, 1997; Schrage, 1993), it can be concluded that prototypes can be useful for eliciting and validating requirements.

Prototyping can help with various agile requirements engineering challenges. A summary of the findings is available below.

**Prototypes can act as documentation.** Many agile projects have decided to replace traditional requirements documents with prototypes as a means for communicating with customer (Ramesh et al., 2010). Prototypes can help the customer trusting the process and doing formal requirements verification (Käpyaho & Kauppinen, 2015).

**Prototypes can help with motivating the team for constant requirements engineering work.** Keeping prototypes up-to-date can be more motivating than updating written documentation. However, prototyping does not solve all the motivation issues that are related to agile requirements engineering work; for example, software developers may lack experience or motivation to do Test-Driven Development or Acceptance Test-Driven Development (Käpyaho & Kauppinen, 2015).

**Prototyping has some effect on improving communication with customer.** When multiple customer groups have different opinions, prototyping can have some effect on finding a commonly accepted opinion. Prototyping can also have some effect on the challenges of getting customer to be present often enough and ensuring that customer has good enough understanding of their role in the project. (Käpyaho & Kauppinen, 2015)

**Prototyping has very little effect on understanding the big picture.** Prototyping does not help understanding the big picture during the early phases of the project or finding out deep system-level problems at the late stages of the project. Prototyping does not have an impact on instability that is

caused by constant requirement changes or on problems with estimating project's schedule or cost. (Käpyaho & Kauppinen, 2015)

**Prototyping does not help with neglected non-functional requirements.** Non-functional requirements are not taken into account properly in agile projects. Prioritization on time-to-market basis can cause challenges with quality. Prototyping does not solve any of these challenges. (Ramesh et al., 2010).

It is worth noting that prototyping cannot solve all the problems that are related to agile requirements engineering. It is often useful to complement prototyping with other practices and methods. To understand business requirements, quality requirements and the big picture, it can be useful to keep track of them with other means (Käpyaho & Kauppinen, 2015).

Based on the case study of Kauppinen & Käpyaho (2015), building acceptance tests based on prototypes can be useful for linking requirements and testing. Similarly, it is reported in a study of *Acceptance-Test-Driven Development (ATDD)* as an agile requirements engineering practices that linking requirements with ATDD can be useful (Haugset & Stålhane, 2012).

Reaching sufficient customer presence can be a problem in agile projects (Ramesh et al., 2010). Having enough quality communication with the customer can be challenging if the customer is not able to participate actively enough in the process. Käpyaho & Kauppinen (2015) suggest that some customer company representatives may need education about agile practices and the responsibilities of e.g. product owner.

It is also good to understand that even though prototyping can help solving some agile requirements engineering challenges, it can also bring some problems of its own to the process. Prototyping can cause non-technical stakeholders to have unrealistic expectations of development speed (Käpyaho & Kauppinen, 2015). Additionally, using prototype code in real software products can be risky (Ramesh et al., 2010).

# Chapter 3 Empirical research

## 3.1 Overview of the project

The empirical research was carried out in an agile software development project, in which I had been involved for more than one year. The purpose of the project was to design and develop a mobile gaming application for Veikkaus, the Finnish national lottery company. Design and development of the product had started in December 2015. During the empirical research phase of the project, my role was to act a user experience designer of the product. Earlier, I had also acted as a software developer in the project. In addition to me, the core team included 4–6 software developers, a scrum master, a product owner, and 1–3 software testing specialists (the number of people fluctuated during the project) during the empirical research phase of the study. Additionally, there were many other stakeholders that were involved in the project, such as user experience specialists and business stakeholders.

## 3.2 Empirical research design overview

The empirical research phase of this study was carried out with two research methods: semi-structured interviews and action research. Method triangulation (i.e. using multiple research methods) was used because if only one method was used and it was based on a biased sample or was analysed incorrectly, the flaw might have stayed completely unnoticed.

Analysis of the data that was gathered from empirical research is described in section 5.1.1 *Analysis process*. Evaluation of the chosen research methods is presented in section 5.1.2 *Method evaluation*.

## 3.3 Interviews

### 3.3.1 Methodology for interviews

Interviews were conducted as semi-structured interviews. According to Wood (1997), semi-structured interviews are suitable for gathering data about expert knowledge. The interviews lasted 44–72 minutes (average interview duration was 59.6 minutes).

Semi-structured interviews proceed by assuming very little about expert's knowledge and use information that the interviewees provide as a foundation for discovering more information about the topic with help of further questioning (Wood, 1997). Before asking specific questions about how the interviewees perceived in the use of prototyping and agile in the project, they were first asked to define these concepts in their own words. This understanding that the interviewees had was used to ask further questions and adjust some interview questions based on the knowledge of each interviewee.

In the interviews, the questions could be asked in a different order, some questions were not asked from all the participants (e.g. if the participant told that they did not have experience of using a specific prototyping approach) and additional questions were asked when it was considered useful. Even though there is a numbering for the interview questions in appendix A, the numbering is only used to make it easier to refer to specific questions in this study and to explain the purpose of asking certain questions. Based on the expertise of each interviewee, each interview was customized so that there was more focus on the topics that the interviewee had deep understanding about and less focus was given to topics that the interviewee was not familiar with.

In the interviews, the interviewees were first asked typical background information, such as age and amount of experience in working in the software industry (this information was asked so that it would be possible to notice if there were some differences between the knowledge of the interviewees based on the amount of experience they had in the software industry).

The interviewees were encouraged to tell their own interpretations about the topics of the interview and they were told that there were no right or wrong answer to the questions.

To put it simply, the main goal of the interviews was to get answers to research questions. The interview questions were crafted so that questions 12 and 20 aim at getting answers to the research questions (they are essentially

the same as the research questions). Question 21 aims at getting more detailed information about research question 2. The other questions serve as introduction to the research problem and research questions.

The default order of the interview questions was chosen so that it would support natural flow of conversation. In the early phases of the interview, there were some easy warm-up questions to get the conversation started. The questions that discussed prototyping (which was a tangible and relatively easy-to-understand topic to the interviwees) were asked first, and the other main theme, agile requirements engineering, which is more abstract topic, was discussed later to avoid overwhelming the participants with lots of questions about an abstract topic in the beginning of the interview. If difficult questions were asked about an abstract topic during the early phases of the interview, the interviewees might have felt uncomfortable during the rest of the interview. Questions 4–12 are related to prototyping and questions 13–21 mainly focus on agile requirements engineering.

Similarly to the example that was described by Wood (1997), the interviews aimed at understanding the current practices that were used in the project (how prototyping and agile requirements engineering were used in the project). Based on the understanding of the current practices that was gathered in the empirical part of the study and the potential improvements to these practices that were discovered in the empirical research and in literature, an improved model of the practices is introduced in section 5.4.2.

The researcher asked feedback about each interview from the participants so that potential flaws in interviews questions could be found and to improve the interview questions if it was discovered that some interview questions might have left room for some improvement.

The interviews were recorded with an audio recorder application and the recordings were archived for the purposes of this study. The researcher agreed not to give access to any third party to the original audio recording files. Notes were written during the interview sessions.

### 3.3.2 Participants

Interview participants included 4 RND Works Oy employees and 1 customer organization (i.e. Veikkaus) employee. All the interviewees were part of a team that was responsible for designing and developing a mobile application for the customer organization. Except for one interview participant, who had

participated in the development of the product since November 2016, all the interviewees had participated in the development or/and design of the product since December 2015. The interviewees had 4–10 years of work experience in the software industry. 100% of the interviewees were male. The average age of the interviewees who wanted to disclose their exact age was 31 years (one interviewee did not want to disclose his exact age).

Because all the participants were from two companies, the generalizability of the empirical part results is not as good as in the case if the research participants were recruited from a higher number of companies. Additionally, all the participants were from Finland, which may make the results less generalizable than in case if the participants represented multiple nationalities.

Before the start of the interview, all the interview participants signed a letter of informed consent (the letter can be found in Appendix B). They were informed about how the data that was collected was used in this study. If an interview participant accidentally revealed some classified project-related information, which could not be published as part of a public Master's thesis document, it was removed from the interview notes.

Additionally, the participants were informed that they had a right to quit the interview at any time if they wanted to do so and they did not have to answer to all the interview questions.

## 3.4 Action research

In addition to interviewing product development team members, another empirical research method was needed to add method triangulation to the study. The chosen method was action research. The main value of action research for this study was making observations about how prototyping and agile requirements engineering were practiced in the organization under study and comparing the results of observations with the results of the interviews. While interviewing is a useful method for gathering data about the researched phenomena, interviewees may not be able to articulate their tacit knowledge about the topic, which may leave some gaps in the data. Observation made it possible to fill the potential gaps in the data that was collected from interviews.

Action research is a suitable research method for investigating complex phenomena in social settings, because reducing social settings for study is

difficult. In action research, a researcher observes a phenomenon and takes part in the researched phenomenon. Typically, action research includes five phases: (1) analysing the current state ("diagnosis"), (2) planning the action, (3) taking the action, (4) evaluation and (5) specifying learning. (Baskerville, 1999) Phases 1–3 are described in this section and phases 4–5 are described in section 4.4 *Action research results*.

**Analysis of the current state:** The project that was being studied was a typical agile software development project in terms of the requirements engineering challenges that the product development team encountered.

While discussing the current state of the project with core team members and observing the project, it became apparent that the team had faced similar requirements engineering challenges that had been described to the typical in agile projects according to academic literature. Namely, the team members reported that the big picture of the project was somewhat ambiguous occasionally, the team had to manage their requirements engineering work with small amount of written documentation, and due to frequent requirement changes and focus on business-critical time-to-market prioritization, it had been challenging to find time for focusing on non-functional requirements (e.g. performance). Additionally, it was mentioned by client organizations' own user experience specialists that the project could benefit from more frequent requirements validation with end users in the form of more frequent usability testing.

**Action planning:** Because prototyping was reported to help with some of the previously mentioned agile requirements engineering challenges based on literature (Käpyaho & Kauppinen, 2015), the researcher wanted to observe the effect of prototyping on agile requirements engineering in the project. Additionally, the researcher wanted to observe if there was any difference in how different fidelity levels (low-fidelity paper prototyping and high-fidelity prototyping) and focus (small focused prototypes vs. large epic-level prototypes) supported agile requirements engineering work.

**Action taking:** This study utilizes observations from creating prototypes for a redesign of a feature of a mobile application (the name of the feature cannot be mentioned in this thesis because the redesigned feature has not been published yet).

Multiple prototypes were created for the redesigned feature. One of them was a low-fidelity paper prototype, which was used in a co-design experiment in a usability test to help target users describe what content they wanted to view within the feature. Additionally, there were three clickable high-fidelity prototypes, which were used for evaluating different ways to interact with the feature on a real smartphone device. One of the high-fidelity prototypes was created in code instead of a prototyping tool because creating a coded prototype made it possible for the end user to rearrange user interface elements in a way that felt realistic from end user's point of view (even though the prototype was created with a different technology as the real product). The other two clickable prototypes were created with a visual user interface prototyping tool. The visual prototyping tool made it easy to and quick to create polished-looking prototypes that felt almost like the real product.

The prototype that was created in code can be called a focused prototype because it only included content reordering and content selection functionality but no other functionality. The other clickable prototypes were large epic-level prototypes. The paper prototype can be called a focused prototype because it focused on content selection.

All the previously mentioned prototypes were used in a usability test session. The main purposes of creating the prototypes were to gather feedback from end users and also facilitate discussion within the development organization about the feature by visualizing various alternative designs. Implementing the prototypes with different approaches (low-fidelity prototype vs. high-fidelity prototype, small focused prototype vs. large epic-level prototype) and tools also made it possible to compare the benefits of drawbacks of the various prototyping approaches.

The author of this thesis created all the previously mentioned prototypes.

# Chapter 4 Results

This chapter includes the results of the empirical part of the study. Sections 4.1–4.3 present interview results and section 4.4 presents action research results.

## 4.1 Agile

### 4.1.1 Definition of agile

The most commonly mentioned aspects of agile were dividing development tasks to short cycles (e.g. sprints in Scrum), doing frequent changes to prioritization of tasks and not having a list of all the requirements in the beginning of a project (unlike in more traditional software development projects that they had done in the past for other companies).

Interviewees I1, I2, I3 and I4 described that in contrast with traditional software development projects, an important part of agile was division of design and development activities to sprints or some other form of cycles. As interviewee I4 described, the goal of this approach was to focus on what makes the biggest benefit for the end user and customer's business needs, doing a small set of features based on those needs and measuring how well the implemented features satisfied those needs. Additionally, he mentioned that it was important that this small set of features formed a meaningful combination. According to interviewee I1, in an agile project, the team and the customer check the outcome gradually, not only after the product is ready. According to interviewee I1, design, development and planning activities have some overlap in terms of scheduling; different activities are not done in separate phases in agile projects.

Interviewee I3 expressed an organizational view of agile development organizations: according to him, *"agility comes from developers being close to*

*designers, stakeholders and end users"*. According to him, efficient agile software development requires that a team (as an organizational unit) can make decisions up to a certain point without having to wait for approval for every small decision. Additionally, interviewee I3 mentioned that in his opinion, agile meant having "low architecture" i.e. not doing lots of upfront planning for software architecture.

Interviewee I1 mentioned that a typical attribute of agile was having little or no documentation, which caused a few challenges that are described in section *4.1.3 Drawbacks of agile*.

Interviewee I1 considered *TDD (Test-Driven Development), BDD (Behaviour-Driven Development)* and *ATDD (Acceptance Test Driven Development)* to be agile practices.

Interviewee I5 mentioned that the definition of agile was somewhat unclear to him; he only mentioned that based on his experience it meant making fast decisions and going forward in the projects based on those decisions.

### 4.1.2 Benefits of agile

Division of the project to small chunks was seen as a positive attribute because it allowed the team to gather feedback from customer and end users at regular intervals. After each short cycle, it was easy to make adjustments to the course of the project if end users or customer felt that some aspects of the product needed improvement. As interviewee I4 mentioned, not doing all the features at once allowed the team to realize earlier during the development of the product if some things that were designed and/or implemented did not make sense. Additionally, short development cycles allowed the team to keep track of the status of the project in small pieces instead of only having a good overall picture of the project at the very end of the project that had happened in some earlier non-agile projects.

Communication inside the team and between the team and the customer was perceived to be better in agile projects. Agile activities, such as Scrum dailies and frequent oral communication made it easy to keep track of the project's progress. Interviewee I2 mentioned that communication depends on the size of the project: if the project is large enough and even if it follows agile methodology, it is difficult to all the project participants to have a clear understanding of the whole project; developers mostly know about the code that they write but not so much about the other aspects of the project, such as

how marketing for the product is done. However, he also mentioned that it was not important for a software developer to know lots of details about e.g. how the daily marketing activities of the product are handled. According to interviewee I2, small software projects and large software projects look different regardless of the project management methodology and the size of the project may have a more significant difference to e.g. communication than whether the project follows agile methodology or not.

Interviewee I5 mentioned that flexibility was a key benefit of agile because it was not necessary to know every requirement and detail at the beginning of the project.

### 4.1.3 Drawbacks of agile

Interviewee I1 reported that having small amounts of documentation caused some inefficiency, because especially some new software developers had lots of questions before they were able to become familiar with the codebase. Due to the small amount of documentation, deep understanding of the codebase and about the products in general is concentrated to a small number of individuals who had been members of the organization for a long time.

Interviewees I1, I3 and I5 mentioned that handling new feature requests and change requests was challenging in agile software development. A large number of various stakeholders made prioritization of features challenging occasionally. Interviewee I1 wondered if there should be more explicit and stricter criteria for deciding, which changes should be accepted. Interviewee I2 mentioned that it was important that there was a dedicated customer representative who would make the final decision about requirement changes.

Interviewee I3 felt that sometimes there was a feeling that the structure of the development process was missing in agile projects (this was a general comment about agile, not specific to the project in which he was working).

Interviewee I1 mentioned that agile software development does not always work optimally, if only some agile practices are followed (e.g. TDD is often used in agile projects so that tests are used as documentation, but if TDD is not practiced, then the need for documentation increases).

According to interviewee I2, some “basic” challenges exist in every software project regardless of project management method, e.g. understanding what is the current overall progress of the project.

Interviewee I2 mentioned that fast-paced agile software development made it challenging to reserve enough time for maintenance of software that was necessary for maintaining the quality of the codebase. Moreover, interviewee I5 said that he did not have enough time for thinking of non-functional requirements. However, they also mentioned that the challenge was not so bad at the time of the interview as it had been in the past. Additionally, they mentioned that due frequent changes to requirements and lack of large milestones that typically exist in traditional waterfall software projects, they had a feeling that software is never "ready".

### 4.1.4 Definition of requirements engineering

Requirements engineering was generally understood as creating some form of specification of what the software product is and what it should do. Requirements engineering work starts when there is discussion about a potential project with a customer. In this phase, it is common according to the interviewees to do a user interface prototype or a technical proof-of-concept-prototype to facilitate discussion about the potential project, its budget and scope. According to interviewee I3, the goal is to provide as compact package (set of features) as possible that satisfies the most critical customer and end user needs instead of promising to implement all the features that all the stakeholders would like to have. In these early phases of the project, prototypes are used to make the discussions less abstract and validate the main assumptions of the customer organization about the product.

During the actual product development project, requirements engineering includes elicitation of feature and change requests, validating if the requests should be implemented and managing the features in the backlog. Validation of requested new features or changed requirements can occur with a prototype (paper prototype or high-fidelity prototype), usability evaluation, technological feasibility evaluation, analytics or A/B tests (or other data-driven approaches). Additionally, validation includes discussions within the core development team, design team and with product owner.

Interviewee I4 had a slightly different view of requirements engineering when compared with the other interviewees. In his opinion, requirements should only contain goals and metrics, not any listings of concrete features. After defining goals and metrics, it is the job of the product team to figure out, which features are needed to reach the commonly agreed goals. According to

this interviewee, creating traditional requirements specifications that contain lists of product features are just waste of time.

### 4.1.5 Definition of agile requirements engineering

The difference between traditional and agile requirements engineering was understood by the interviewees so that agile does not require specifying all the features of the product upfront in the beginning of the project. New requirements and changes to existing requirements can emerge at any given time during the project.

In agile requirements engineering, features are prioritized frequently and iteratively during the project (usually in sprint planning sessions or Scrum dailies). Product owner has an important role in the prioritization process as a decision maker.

To make agile requirements engineering work well, it was suggested by interviewee I4 that the whole team should participate actively in validating new requirements or changes to existing requirements. In his experience, it is possible that if only one person (for example, a product owner) is active in the validation process, the outcome may be implementing more features than it is necessary to reach the goals of the project. For this reason, he stressed out that the whole team should have very clear understanding of the goals and metrics of the project. Unfortunately, based on his experience on working for various companies, defining explicit goals and metrics is usually difficult at most companies. Interviewee I1 commented that the most efficient way to validate requirements is to create a prototype or some other model of the requirements and ask feedback from the person who requested the feature (and potentially from end users). The person who requested the feature is usually a domain expert and can clarify the requirements if it is clear based on the prototype or model that the requirement was misunderstood or if some important details are missing.

Interviewee I1 commented that requirements should be ideally stored in a backlog (either in a digital backlog tool or on a physical wall of Post-It notes). He mentioned that written requirement documents will be out of date as soon as they are written and therefore they are not optimal for agile projects, in which requirements change frequently.

### 4.1.6 Requirements engineering in our project

Interviewees commented that requirements usually came from various stakeholders. Sometimes, new requirements or changes to existing requirements were added based on customer feedback, analytics or core development teams' own ideas.

Developers handled the technical side of requirements validation, and user experience designers and business stakeholders evaluated the impact of the requirements to user experience and business. For large features, the product owner made the final decisions about whether the new feature or change to an existing feature was made. For bug fixes and very small changes, the core team usually made the decision if the feature or change was made and when it was made.

Due to the large size of the customer organization and small size of the core development team, requirements prioritization was very important because the team received feature requests from a large number of stakeholders. Interviewee I2 expressed that it was good to have a product owner who had enough decision-making power in the organization to do the prioritization without having to ask for approval from other people.

There were two options for storing requirements: an electronic backlog tool and a physical wall of notes. Both of them were in active use, although some members of the team preferred to use the physical wall of Post-It notes because the physical wall was faster to use when making small changes.

## 4.2 Prototyping

### 4.2.1 Definition of prototyping

According to the interviewees, the main purpose of prototyping is to test quickly if a suggested idea makes sense before creating the real product or feature, which takes more time and costs more money than a prototype. Interviewee I2 commented that prototypes should be created so that they can be thrown away; creating the prototype should take a minimal amount of time and no emotional bond should be left to the throwaway prototype. He added that prototypes can reveal if the idea works at all. Interviewees I1, I2, I3 commented that it is also possible to use prototypes to explore multiple alternative solutions and evaluate which alternative (or a combination of some elements from multiple alternatives) works best.

Interviewees I3 and I4 stressed out that a prototype should focus on the most important features and user paths in the product; they thought that at least in the early stages of new product development process it is crucial to understand what is the minimum number of features that are truly necessary. Interviewees I3, I4 and I5 commented that prototypes could be used for estimating budget before making commitment to developing a product or feature.

### 4.2.2 Interviewees' previous experience with prototyping

The definitions that the interviewees had for prototyping varied slightly based on their experience with prototyping. Interviewee I4 only talked about user interface prototypes. The other interviewees had more experience with technical proof-of-concept prototypes, but they were also familiar with user interface prototypes and they had created some user interface prototypes in code.

Interviewee I4 told that prototypes were used at the beginning of every large project and when making significant changes to existing products. Prototypes were not used for small incremental changes or when making changes to features on less frequently used user paths.

Interviewees I1, I2 and I3 mentioned that prototypes had been used for sales and marketing. Interviewees I1, I3 and I4 mentioned that they had used

prototypes for requirements elicitation and validation. Validation was done in usability tests and/or by asking feedback from customer.

All the interviewees except I4 had used prototypes to validate assumptions about technical feasibility of new technologies by creating technical proof-of-concept prototypes.

### 4.2.3 Benefits of prototyping

All the interview participants mentioned that prototyping is fast and cheap when compared with creating the real product first and realising only after that that there was some problem with the product (e.g. a usability problem or if the product consisted of features that were completely unnecessary to end users). Interviewee I3 mentioned that it was really eye-opening to see in usability tests that two almost identical user interfaces can lead to two completely different end results.

Interviewee I4 commented that prototyping can bring the whole team together, including designers, software developers and product owner, so that they can understand what the team is supposed to accomplish together.

Interviewees I1, I2, I3 and I5 said that proof-of-concept prototypes had been helpful for testing if a certain technology was ready for actual production usage and for evaluating what introduction of a new technology requires from stakeholders, such as *DevOps (Development and Operations)* or *QA (Quality Assurance)* specialists.

Interviewee I1 had some recent experience of using prototyping as ideation tool. Usage of prototyping during development of a new product had yielded a completely different product than what the initial thoughts of the customer about the product had been.

Interviewee I2 mentioned that prototyping made it easier to gather feedback about the product from real end users; otherwise end users' point of view might not be taken into account so strongly.

When a new product is implemented with new technologies, it can take a while before the development team can get the development process up and running. Interviewee I2 commented that creating a prototype with a design tool can help the project to a fast start. Additionally, he mentioned that creating prototypes with design tools helps non-coders discuss the product with developers and make their ideas less abstract and closer to how the product works in reality.

All the interviewees mentioned that prototypes can help estimating the amount of work that is required for the actual product.

### 4.2.4 Drawbacks of prototyping

Interviewees I2, I3 and I4 commented that prototypes are easy to fall in love with and all the stakeholders may not understand that prototypes are usually supposed to be thrown away. Interviewee I4 commented that sometimes some people are not willing to throw away a fancy prototype even if it does not work well (e.g. it has usability problems or it is not technically or financially feasible). Additionally, all the interviewees commented that polished high-fidelity prototypes can create unrealistic expectations especially to non-technical stakeholders. Therefore, it is important to communicate that a prototype is a throwaway version of the product and it will take more time to implement the real product.

Interviewee I2 commented that it is possible for designers to create technically infeasible prototypes with design tools. Therefore, a designer must take into account the limitations of the target platform when creating prototypes.

According to interviewee I4, everything cannot be tested with a prototype or if it is implemented with such technologies (e.g. coded HTML prototypes) that it is possible to prototype the small nuances and microinteractions of the product, the prototype will probably take such a long time to make that the benefit of saving time and money with prototyping is lost.

Products that display lots of personal user-specific data and dynamic user interfaces that change their state based on user input are usually difficult to prototype with design tools. Interviewee I4 mentioned that using mock data can make the prototypes feel unrealistic in usability tests because average end users do not see a difference between content and the user interface; if the content is unrealistic, the feeling of simulating real product usage with a prototype is lost.

Interviewee I5 commented that because prototypes are usually created quickly to save time, usage of prototype may lead to introduction of new bad practices because the prototypes were created in a hurry. Additionally, he mentioned that the drawback of coded prototypes is the temptation to use prototype code in a real product to save time (even though prototype code will be most likely more difficult to maintain).

### 4.2.5 Prototype fidelity

Interviewees I3 and I4 were most positive about using paper prototyping as part of a design process. The other interviewees felt that clickable prototypes were easier to understand and they were hesitant about the idea of showing paper prototypes to end users. Specifically, interviewee I5 said that it was difficult to understand how the flow of the user interface and how different screens are related to each other from the paper prototypes. However, interviewees I1 and I5 said that paper prototypes could be useful as product development organization's internal tool for discussing the requirements. The difference between the opinions of I3 and I4 and the rest of the interviewees can most likely be explained by the fact that I3 and I4 had more experience with paper prototyping than the other interviewees.

Interviewees I3 and I4 said that there was a mostly linear timeline for the usefulness of different prototype fidelities; low-fidelity prototypes were more useful in the early stages of new product development, whereas high-fidelity prototypes were more suitable for later stages of product development process. Interviewee I3 considered that the best usage situation for paper prototypes was concept design phase of a new product. Interviewee I4 commented that paper prototyping becomes difficult as soon as the prototype needs to contain any interaction. Interviewee I3 thought that high-fidelity prototypes would be more suitable when modifying existing features, because earlier versions of the design are already available in digital format and modifying them will probably take less time than creating a new low-fidelity prototype from scratch.

Despite having a positive attitude towards low-fidelity prototypes, I3 argued that because using a high-fidelity prototype feels almost the same as using a real product, it is better to use high-fidelity prototypes if there is enough time to create them.

### 4.2.6 Small focused prototypes vs. large prototypes

The interviewees were asked about how they felt about the difference between small focused prototypes and large prototypes. In this context, a focused prototype means a prototype that presents a single feature or component or some other small part of the user interface. A large prototype means a prototype that presents a whole product or an epic.

Interviewees I3, I4 and I5 emphasized that it was important to understand the big picture first and only after that spend time on refining smaller pieces of the user interface if necessary. If small parts of the user interface were iterated as small focused prototypes, it was considered important to only focus on the most important choices that the users have to make and the most frequently used paths and not spend time on refining unimportant things. Interviewee I3 took the discussion to a more fundamental level: if it is detected that there is a problem with a certain user interface element, perhaps it is useful to consider if the element should exist at all or if it should be located in a different place instead of trying to improve the design of a problematic element by iterating the design.

Interviewee I4 mentioned that iterating a small piece of a user interface separately can be useful if it is understood based on usability tests, customer feedback, analytics or some other means that there is a problem in a certain small piece of the user interface. After iterating the small piece of a user interface as a focused prototype, it was considered important to bring it back to the large user interface prototype so that it was easy to use it in usability tests.

Software developers commented that because user interfaces are modelled as components in code, it would be best to handoff prototypes to developers as focused component-level prototypes so that they can understand more easily how each component should work. However, it is also necessary to have an image or a large prototype that shows where the component is located in the user interface.

### 4.2.7 Prototyping for different audiences

The interviewees were asked how they perceived the role and suitability of the prototype when it was used product development organizations' internal purposes (e.g. discussing new features within the team and with stakeholders or creating a design handoff from designers to developers) or external purposes (e.g. usability testing with end users).

Interviewees I1, I2 and I5 would have preferred to only show high-fidelity prototypes to end users because they felt that paper prototypes were more difficult to understand. Interviewees I3 and I4 felt that it was alright to show paper prototypes to end users, even though they mentioned that paper prototypes have many limitations and they work best during the early phases

of the design process. Interviewee I4 emphasized that regardless of prototype fidelity, a prototype should not be iterated inside the development organization for extended periods of time and it would be necessary to receive frequent feedback from end users. For development organization's internal communication about features and the overall structure of the user interface, it was acceptable to use a paper prototype according to all the interviewees.

According to the interviewees, focused prototypes were suitable for team's internal discussions and stakeholder communication, but they preferred large (product-level or epic-level) prototypes for usability testing because they thought that it would be difficult for an outsider to understand a small focused prototype because end users would not know the context of feature-level or component-level prototypes unlike the insiders of the organization.

For design handoffs from designers to software developers, the interviewed software developers preferred small component-level prototypes, even though they said that it would be important to have another prototype or picture that shows the context in which the component is used.

For sales and marketing purposes, the interviewees who mentioned this usage purpose of prototypes preferred clickable product-level high-fidelity prototypes for the same reasons why some interviewees preferred high-fidelity prototypes and product-level or epic-level prototypes for usability testing with end users.

## 4.3 Agile requirements engineering with prototyping

### 4.3.1 Requirements of fast-paced agile development for prototyping

In fast-paced agile software development, some challenges are caused by both the fast pace and agile. According to interviewees I1, I3 and I4, fast pace in a project typically simply limits the number of prototyping rounds. Additionally, interviewees I1 and I3 mentioned that fast-paced projects are challenging for from the point of view of prototyping and other forms of experimentation, because time pressure may lead a development team to discard the most experimental and risky ideas to maintain the velocity of the project, even though the ideas that have high risks typically have high rewards if the idea is successfully implemented.

Interviewee I1 mentioned that agile method Scrum, which emphasizes strict time-boxing of development sprints and strict prioritization of features makes doing experiments difficult because it can be difficult to evaluate time that an experiment requires before making the experiment.

Frequently changing requirements in agile projects cause a need for frequent reviewing of the changed requirements. Interviewees I1, I3 and I4 mentioned that prototyping should be done more frequently if it was supposed to be used as a tool for evaluating the new requirements. Interviewee I4 stated that if a prototyping tool does not have good support for keeping the prototype up-to-date easily after requirement changes, it can be time-consuming and frustrating to update the prototype. If prototypes are not up to date, they cannot be used as documentation for developers and stakeholders.

Due to small amount of written documentation in agile projects, the prototypes need to describe functionality very clearly if they are used as a replacement for written documentation. Interviewees I1, I2 and I5 preferred to view high-fidelity prototypes because high-fidelity prototypes are clearer and less ambiguous than paper prototypes.

Understanding big picture can be challenging in agile projects. Due to this reason, multiple interviewees wanted to have some large product-level prototypes to understand the big picture of the product.

### 4.3.2 How to improve prototyping for fast-paced agile software development

In order to be useful for fast-paced agile projects, prototyping needs to adapt so that prototypes are fast to create and easy to modify. Interviewee I4 argued that it is important to create lots of prototypes fast because it makes it possible to have frequent contact with end users and receive frequent feedback about the product from them. If time constraints limit the number of prototypes that can be created, interviewee I4 suggested that it would be best to focus on those paths in the user interface that are interacted by a really high percentage of end users. Interviewees I1 and I2 argued that it was important to do prototyping and usability testing frequently because doing it rarely may result in discarding the most experimental and risky ideas (that typically have high rewards when successfully implemented when compared with "safe" ideas) without even trying if those ideas could work.

An improvement that was mentioned while discussing prototyping with interviewees I3, I4 and I5 was selecting a clear focus for a prototype. Interviewee I4 stated that if it was clearly understood that there was a problem with a certain piece of a prototype, it could be useful to iterate the design of that piece of the user interface as a small focused prototype to save time. Interviewees I3 and I5 mentioned that it would be more convenient for software developers to receive design handoffs from designers as small component-level prototypes instead of large product-level or epic-level prototypes because user interfaces are usually modelled as components in code. However, they also mentioned that design handoffs should also include some sort of explanation of how the components are used as part of the user interface (e.g. a picture of the user interface and a flowchart of different user interface states). Interviewee I5 commented that it would be good if each prototype would focus on one thing only so that it would be faster to create the prototype and easier to understand it.

Prototyping user interfaces that contain user-specific personal data was discovered to be slow and cumbersome according to interviewee I4. Moreover, interviewee I4 reported that using mock data in prototypes caused challenges in usability tests because users felt that the prototypes that contained mock data that was not personalised made the experience of using a prototype not feel realistic enough. If it was necessary to create prototypes for multiple user segments for usability tests to make the experience feel more realistic, it

required to create one prototype per user segment with segment-specific mock data for each segment. Needless to say, creating multiple versions of the same prototype with different content was considered unsuitable for fast-paced agile projects with strict time-boxing of design tasks. Interviewee I2 suggested that there should be a lightweight development environment that could be used for creating coded prototypes without all the complex logic code that is needed for real production software.

One way of tackling the challenges of agile requirements engineering that came up in interviews was creating simpler and lower fidelity prototypes. Keeping detailed high-fidelity prototypes up-to-date after frequent requirement changes was considered a challenge by interviewee I4. Simpler prototypes that would not contain such many details would be easier to keep up-to-date. Additionally, low-fidelity prototypes would be faster and cheaper to create. Moreover, all the interviewees mentioned that polished high-fidelity created unrealistic expectations of development speed, which could cause even more time pressure on the team than it was normal in a fast-paced agile project. Interviewee I4 suggested that prototyping and user experience design should be adjusted to a more lightweight format for fast-paced agile projects; for example, the evaluation of prototypes with end users could be done as lightweight "guerrilla usability testing" instead of as traditional usability tests at a usability laboratory.

Because information is often exchanged through oral communication in agile projects, it is necessary to involve the relevant people in the prototyping process so that they understand why each prototype is created and what design problems it aims to solve. Interviewee I3 argued that there should be at least one software developer involved at each design sprint or workshop where prototypes are created so that developers can understand the purpose of the prototypes and they can give their own input to the prototyping process.

### 4.3.3 How prototyping impacts agile requirements engineering

The role of prototypes in the process of eliciting and refining requirements was considered important by interviewees I1, I3 and I4. They also mentioned that prototypes were good for making stakeholders' abstract ideas more concrete and facilitated discussion about the ideas. Additionally, interviewee I3 mentioned that prototypes were effective in removing stakeholders' potential fears and misunderstandings about if the features that they requested were not

going to be implemented in a way that they had hoped. Moreover, interviewees I1, I3 and I4 mentioned that prototypes made it easier to evaluate suggested requirement changes by making it easy to do usability testing and gather feedback from domain experts within the product development organization. Interviewees I2, I3 and I4 mentioned that prototypes could be used as a risk management tool by making it possible to understand problems in a suggested requirement change before commitment was made to spend lots of time and money for developing a feature. In the early phases of product design, prototypes could also have a more fundamental effect in validating or refuting fundamental assumptions about what the product should be all about.

Interviewee I4 commented that in his experience, prototyping was a useful practice for requirements engineering because if prototypes were created and evaluated with end users and within the product development organization, a fewer number of unnecessary features were implemented. Not doing prototyping usually resulted in developing an excessive number of useless features without critical evaluation of the features.

Several interviewees mentioned that prototypes could be used as requirements documentation in place of traditional written requirements documents; they thought that prototypes were useful for documenting user-facing features. However, prototypes did not support understanding and documenting all the non-functional requirements.

In interviewee I3's opinion, prototypes (regardless of being created with a design tool or in code) could help developers write less buggy code because prototypes helped developers have a clearer understanding of requirements: what was supposed to be included in a feature and what should not be included. Additionally, interviewees I3 and I5 commented that large product-level or epic-level prototypes could help developers forecast upcoming features and choose adequate system architecture for future needs.

Interviewees I2 and I3 mentioned that in addition to facilitating discussion about user interface, prototypes were useful for evaluating technical feasibility of ideas, if the prototypes were implemented as coded prototypes with same technology stack as the actual product. Additionally, prototypes could help assessing the effect of new features to the technical aspects of the product, such as performance.

When comparing requirements engineering with or without prototyping, interviewee I5 mentioned that prototypes made it easier for software developers to understand user interface requirements. He said that prototypes

were better for explaining requirements than static images or written documents because prototypes made it easier to understand the flow of the user interface and see how different screens were related to each other. Interviewee I5 mentioned that the way how prototypes are implemented is crucial for creating "self-documenting" prototypes: he felt that clickable high-fidelity prototypes (regardless of being implemented with a design tool or in code) were best for explaining requirements; in his opinion, paper prototypes were more difficult to understand and did not explain the flow of the interface adequately.

### 4.3.4 How prototyping helps with agile requirements engineering challenges

**Little documentation:** Prototyping can help with challenges that are caused by only having small amounts of documentation in agile projects. Interviewees I2, I3, I4 and I5 stated that prototypes can act as documentation. Interviewee I4 commented that prototypes can be very detailed and they can be more useful documentation for the user interface than written requirements specifications; according to him, written documents are not very useful for describing how a user interface should work.

Interviewee I3 commented that if prototypes are used for specifying requirements, it can be problematic sometimes because changing the prototype later may require re-negotiation of requirements. Avoiding making the prototypes very detailed can make this problem less prominent, but lack of detail can also make the prototype so ambiguous that some people misunderstand the requirements.

According to interviewee I2, the usefulness of the prototype as a tool for documenting requirements depends on who creates the prototype. If the prototype is created by a user experience specialist, it can document the requirements properly. The reasoning behind his opinion is that because user experience specialists know end users' needs and stakeholders' needs well and because they have thorough understanding (from studies and experience) of how different design choices affect usability and other relevant issues, the prototypes that they create can act as useful documentation of software's user-facing features. He mentioned that an average programmer does not have such a broad understanding of different factors that affect the requirements and therefore prototypes that are created by programmers are usually not useful

for documenting requirements. However, he stated that software developers could create technical proof-of-concept prototypes that can document technological requirements of a software product. However, he mentioned that for documenting technical requirements traditional software documentation is probably more useful and some traditional software documentation needs to be created anyway for making software development work convenient.

Interviewee I5 pointed out that if a prototype is used as user interface documentation, a clickable prototype will be preferable because it shows the flow of the user interface and how different screens are related more clearly than a paper prototype. Instead of using prototypes as user interface documentation, slideshows or screen compositions could serve a similar purpose in his opinion.

**Motivation issues to do constant requirements engineering work:** Generally speaking, the effect of prototyping towards motivation issues to doing constant requirements engineering work received mixed reactions from the interviewees.

Interviewees I3, I4 and I5 stated that prototypes can help making it clear which requirements have changed, provided that the prototypes are always kept up-to-date. If the prototypes are out of date, they can make the issue worse by causing confusion.

Interviewee I3 mentioned that creating prototypes frequently and versioning the prototypes can make it easier to trace when a certain change was made and thus make it easier to notice when some problematic change to requirements was made in the past. Still, because prototypes are not self-explanatory, prototypes are not very useful for explaining why a certain change was made. Interviewees I1 and I3 commented that the most significant reason that caused loss of motivation towards requirements engineering work is if the reasons behind changes are not explained properly.

Interviewee I1 mentioned that making frequent changes to requirements can be tiring in agile projects sometimes, but experimenting with different ideas by prototyping is good because by making experiments and changing the requirements based on experiments the product gets better over time and the team can also learn something during the process of experimentation.

Interviewee I2 commented that to keep the motivation to doing constant requirements engineering work it is important to make the prototypes so fast that having to throw away a prototype does not feel bad. If prototypes take too

much effort and time to make, having to throw away a prototype due to changed requirements can affect motivation negatively.

**Achieving quality communication with customer:** According to interviewees I1 and I4, prototyping can help communication with customer and various stakeholders by making abstract ideas more concrete. If a prototype can be shared in electronic format (e.g. via a link that can be opened in a web browser or mobile app), prototype can help busy customer representatives keeping track of the project's progress and make it easy to them to give feedback about the product even if they do not have possibility to attend all the regular meetings with the product development team. Still, it is useful to have some customer representatives that can participate in the daily activities of the team and prototypes cannot replace these face-to-face communication activities completely; active customer's participation is necessary in a successful agile software development project.

Interviewee I3 commented that prototypes can facilitate discussions with some stakeholders who may have an abstract view of the product. However, he commented that if a communication challenge is caused by lack of time to meet team members and discuss the requirements with them, prototyping is quite heavy solution to the problem (because making prototypes can take a long time sometimes) and other more lightweight solutions could be more applicable for facilitating communication.

Interviewee I5 had a more pessimistic view of the topic and he felt that prototypes were not very useful for overcoming potential communication challenges in agile projects.

**Missing big picture:** If prototypes are done as large product-level prototypes or epic-level prototypes, prototyping can help understanding the product-level big picture. Unfortunately, these product-level and epic-level prototypes do not help creating a deep understanding of how the product is linked to company's strategy or vision. However, interviewee I1 pointed out that prototyping could have some effect on making vision less abstract by framing the vision to a concrete product.

According to interviewees I2 and I3, prototyping is especially useful at the beginning of the development process of a new product. Additionally, prototyping can help the team understanding what the product could be in the future even if the ideas that are portrayed by the prototype might not be

feasible in the short-term future. According to interviewees I2, I3 and I5, prototypes that help forecasting the future could be useful for software architecture design, provided that the prototypes are broad enough so that they show the whole product or at least the most important parts of it so that the overall structure of the product is clear. However, because frequent changes to requirements are common in agile projects, forecasting the long-term future can be challenging, as it was mentioned by interviewee I4.

According to interviewee I1, it can be challenging to understand how individual feature requests affect the big picture of the product. He mentioned that adding these individual features to a prototype that contains the whole product could help the team understand the effect of individual changes to the big picture better.

Interviewee I4 stated that prototyping has significant impact to cost and schedule estimation because it is easy to see from the prototype e.g. how many views the user interface contains, which components are needed and how complex logic is needed for keeping track of the state of the user interface. If requirements specification is done as a traditional list of features, it is difficult to guess how much time is needed to implement each feature because a written specification does not tell how complex the user interface is.

Interviewee I1 commented that experimentation-driven iterative agile development that includes prototyping will most likely result in a different type of technical implementation than a more plan-driven approach, but he also mentioned that it is not possible to know all the requirements beforehand in a typical software project. Prototyping can at least help anticipating some upcoming changes to make it easier to maintain good system architecture.

Interviewee I4 mentioned that before prototyping can help understanding big picture, it is always necessary to define the goal of the product. Prototyping and other similar practices are simply just means to get towards the goal.

**Not enough emphasis on non-functional requirements:** Prototyping can have some effect on understanding non-functional requirements. For example, interviewees I3 and I5 said that user interface prototypes help thinking of performance requirements. Additionally, he mentioned that technical proof-of-concept prototypes can help exploring new good practices for e.g. security.

However, the effect of prototyping on taking non-functional requirements into account is limited. Interviewees I2 and I4 stated that the effect is small or at least not very obvious.

**Challenges with prototyping itself:** Risks of using prototype code in production software were well known by all the interviewees and they had gained some experience with the problem in the past. The interviewees commented that maintenance of prototype code was typically more difficult than maintenance of normal production code. The root causes of the problem were described as insufficient communication about the purpose of the prototype (it was supposed to be a temporary experiment) and tight schedules in fast-paced software projects. Interviewee I4 commented that the same issue is had occurred when doing some A/B tests (test code might not have been cleaned from the codebase after A/B tests were over). Interviewee I5 suggested that the risk of using prototype code in production could be mitigated by creating prototypes with design tools instead of in code so that it would be impossible to reuse any parts of the prototype in real production software.

The challenge of creating unrealistic expectations of development speed was acknowledged by all the interviewees. Just like the problem with using prototype code in production software, the issue with unrealistic expectations was mainly seen as a communication problem. To mitigate the issue, interviewees I1, I4 and I5 suggested adding some visual cues to the prototype to make it clear that the product is far from ready. Interviewee I5 believed that creating the prototypes with a design tool would make it clear that the prototype is only a drawing and the actual implementation of the product will take much longer time. In contrast, interviewee I2 believed that user interface design tools that allow creating finished-looking prototypes quickly are more likely to create unrealistic expectations of development speed than coded prototypes.

Interviewee I4 mentioned that polished prototypes can make users censor their honest opinions in usability tests, which will result in unrealistically positive usability test results.

### Summary of how prototyping helps with agile requirements engineering challenges according to interviewees' opinions

*Table 4.1:* Effect of prototyping on agile requirements engineering challenges.

| Agile requirements engineering challenge | Effect of prototyping |
|---|---|
| Little documentation | Prototypes can act as user interface documentation. |
| Motivation issues to do constant requirements engineering work | Prototyping can have some effect but cannot solve the issue completely. To be useful for motivating the team for constant requirements engineering work, prototypes themselves need to be kept up-to-date. |
| Achieving quality communication with customer | Prototyping can have some effect on this issue but prototyping cannot solve all the potential communication issues. |
| Missing big picture | Large product-level or epic-level prototypes can help understanding the product-level big picture. However, prototypes are not effective in explaining how the product is aligned with company's strategy and vision. |
| Not enough emphasis on non-functional requirements | Prototyping can help with some non-functional requirements because it makes it clear to developers how the product is supposed to work and after knowing how the product works it is easier to focus on non-functional requirements. However, prototyping does not solve this challenge completely. |
| Challenges with prototyping itself | Risks of using prototype code in production and creating unrealistic expectations with prototypes are understood by the team based on previous experience. To solve these issues, good communication is needed to explain the purpose of prototyping. |

## 4.4 Action research results

The results of action research are reported here according to the Action Research Cycle that is described in Baskerville's (1999) article.

**Evaluation:** When comparing the different prototyping approaches that were used in the project (high-fidelity prototyping vs. low-fidelity prototyping, small focused prototypes vs. large epic-level prototypes), it became apparent that there were upsides and downsides in each approach. The low-fidelity prototype was the fastest of all the prototypes to create and it allowed the end users to participate in ideation of how the redesigned feature could work without requiring any previous experience of prototyping. Therefore, one might think that the paper prototype was most suitable for the purposes of a fast-paced agile software project solely based on how fast it was to create. However, the limits of the paper prototype were reached quickly when there was a need to simulate complex user interactions and microinteractions. Clickable prototypes were considered more useful for those purposes by creator of the prototypes and the team members. Additionally, clickable prototypes could include some attributes that were difficult to implement in a paper prototype, such as animations and view transitions. Additionally, the possibility to share the clickable prototypes via a link that could be opened in a web browser or a mobile app was considered useful by the team members. To interact with the paper prototype, it was necessary for the team members to visit the same room where the prototype was located.

Large epic-level clickable prototypes were considered useful for understanding the redesign of the feature as a whole and getting the big picture of the epic's overall future direction. Unfortunately, the large prototypes that contained all the important user-facing features of the epic were slow to modify when team members and the creator of the prototype came up with new ideas of how the feature could work. Therefore, it can be said that these large prototypes were not ideal for a fast-moving agile project, in which requirements changed frequently. The problem was most significant in those clickable prototypes that were created with a design tool. The coded prototype was slightly easier to modify afterwards; for example, rearranging the user interface elements to a new order was a matter of changing the order of function calls in code and removing a user interface element that appeared in the middle of the prototype was just a matter of deleting some code.

Additionally, if there would have been a need to customize the contents of the prototype for different end user segments, it would have been quite easy.

The reason why modifying the clickable prototypes with the design tool was slow was that the tool required the designer to draw all the combinations of the different user interface states that the prototype was supposed to simulate. Because the prototypes consisted of a quite large number of user interface elements that could appear in almost any order, creating the prototypes was quite slow. Additionally, if a change to one user interface component was made with a drawing tool, the change had to be uploaded manually to all the screens that contained the component (even if the updated components could be uploaded to the prototyping tool with a plugin of the drawing tool, the process was quite time-consuming). However, the most difficult problem with the prototyping tool was that the touch targets on each screen were linked to the coordinates of the user interface elements, not to the elements themselves. When a user interface element that appeared in the middle of the prototype was deleted, all the touch targets for the elements that appeared below the removed element had to be moved manually to their new vertical position. Due to the time pressure and frequent requirement changes in in fast-paced agile projects, the creator of the prototypes felt that creating large prototypes with lots of detail that could get soon out of date was a bit risky.

Customizing the clickable prototypes for different user segments with the design tool would have required creating multiple copies of the same prototype and modifying them. This was impractical for an agile project, in which requirements changed frequently and updating the prototypes after a requirement change would have required updating all the copies of the prototype manually.

To sum up, creating the clickable prototypes quickly (to make high-fidelity prototyping more suitable for a fast-paced agile project) would have required a prototyping tool that would have supported a component-based model of creating user interfaces and some automation for time-consuming routine tasks.

Regardless of the downsides of high-fidelity prototyping, it was possible to save time in high-fidelity prototyping by using earlier created versions of a prototype as a basis for new versions (with paper prototypes, creating two different versions of the same prototype would have required duplicate work when compared with creating just one paper prototype.

The focused prototype that was designed to simulate a small number of specific interactions was faster to create than the large epic-level prototypes. Additionally, it was slightly more immune to frequent requirement changes in an agile project because the prototype only focused on a small part of the user interface and if changes occurred in some other parts of the user interface, the changes did not need to be updated to this prototype because the features did not exist in this prototype. Therefore, it can be said that small focused prototypes are suitable for agile fast-paced software projects, in which changes happen frequently. Because the focused prototype did not take so much time to create as large epic-level prototype, the creator of the prototype did not feel that lots of effort was wasted if the prototype had to be thrown away.

Even though focused prototypes have benefits when considering their suitability for supporting agile requirements engineering, it is worth noting that at least in the case of the prototype in this study, it was also necessary to create at least one large epic-level prototype so that end users and the team could understand the big picture of the epic.

Creating multiple prototypes for one usability test session i.e. applying parallel experimentation was considered useful because it enabled exploring multiple design alternatives at once. Each prototype had some strengths and weaknesses from end user's point of view and evaluating all the prototypes at once made it easier for the team to understand the upsides and downsides of each option and make informed decisions for further refinement of the feature.

During the process of prototype creation and validation, the prototypes facilitated discussing several potential design alternatives within the team and helped the team understand the desired future direction of the epic. Because the prototypes only focused on one epic, the prototypes did not help understanding wider product-level big picture or how the epic was related to strategy or vision. Anyway, regardless of how well the big picture was understood, the prototype was considered as useful user interface documentation by designers and developers.

Because the prototypes were evaluated with end users, the team seemed to be motivated to consider making the changes that were suggested by the prototype. However, it is possible that the motivation was not affected by the prototype itself; some other practice that would have involved end users in the design process might have had the same impact.

As it was believed when planning to start doing more prototyping, addition of prototyping resulted in more frequent validation of requirements with end

users than previously. It is possible that end user involvement could have happened without prototyping with the help of some other practices, but it can be said that prototyping provided a natural way to engage end users in requirements validation. Additionally, prototyping acted as a way to add slightly more experimentation to the user experience design process.

Even though the prototypes did not explain directly any requirements that were related to other systems than the one that the team was developing itself, the team understood quickly that implementing all the functionality of the prototype would have required some changes to backend software that were not feasible in the short-term future. Therefore, creating the prototype before starting to do the technical functionality of the epic may have saved the team some effort by helping the team understand the requirements that the epic had to external systems.

The potential challenge of using prototype code in production software did not happen in the case of this study, neither did the prototype cause any unrealistic estimates of development speed for the feature because the purpose of the prototypes was communicated clearly.

**Specifying first cycle of learning:** The prototype was created by one user experience designer from start to end. It was realised while discussing the prototype with the team that a few other team members would have had some ideas that could have been useful for the prototype. Unfortunately, many team members were quite busy at the time when the prototype was created and the prototype was created by only one person to save time.

To improve prototyping, it would be useful to involve multiple team members and other stakeholders in the prototyping process. Because agile software projects do not usually have extensive requirements documentation, it is unlikely that one designer will know all the requirements independently. Therefore, it would be useful to do some form of participatory design during the design process. This could happen as a collaborative sketching workshop and ideation session at the beginning of the design process.

Because detailed prototypes are quite difficult to maintain in an agile project, in which requirements changes frequently, it could be useful to do simplified prototypes that do not have such a high level of detail.

**Summary:** Prototyping can be improved for fast-paced agile software development by using more simplified prototypes, focused small prototypes

and by carefully considering when to use each prototype fidelity. Additionally, it is useful to involve multiple team members and stakeholders in the prototyping process. Making high-fidelity prototyping more suitable for fast-paced agile projects requires better tooling. Testing multiple designs in one usability test session can help evaluating multiple alternative design directions at a rapid pace.

Prototyping can support agile requirements engineering by acting as documentation, facilitating communication and motivating the team for constant requirements engineering work. Prototyping can also have some effect on understanding product-level big picture.

# Chapter 5 Discussion

## 5.1 Methodology for analysis

This section describes how data was analysed. Because it is also important to understand the limitations of the used methods, an evaluation of the methods that were used in this study is presented in section 5.1.2.

### 5.1.1 Analysis process

Analysis was done in two phases:

The first phase was done using bottom-up approach that consisted of labelling concepts in interview notes and observation notes and categorizing the concepts. During this phase, the notes were read through and the interview recordings were listened to multiple times. While reading the notes and listening to the recordings, concepts that were considered important were added to Post-It notes. Those Post-It notes that listed a similar concept were grouped together. Additionally, when it was discovered that there was a link between categories, the link was drawn between the categories. The categories were not predefined before the analysis so that it would be possible to notice if unanticipated categories emerged from data.

As a whole, the first phase of the analysis process was similar to using grounded theory. Grounded theory is a qualitative research method, in which a theory is formed from concepts that are present repeatedly (or absent to a great extent contrary to expectations) in interviews or observations to create a theory that is grounded in reality (Corbin & Strauss, 1990).

The second phase was analysing the data with top-down approach, using research questions and literature review as a basis of analysis. The data from the empirical part of the study was compared with the findings from literature.

### 5.1.2 Method evaluation

Method triangulation (i.e. using multiple research methods) was used to improve the quality of the research; if one research method gave biased data, it would be possible to notice that the potentially biased data was not similar to the data that was gathered with the other research methods.

Evaluation of the research methods that were used for literature review and empirical research is available below:

**Literature review:** Section 2.1 of this thesis defines the data collection and literature selection criteria clearly. The thesis followed a modified (less strict) version of systematic literature review guidelines that were described in Staples & Niazi's study (2007). In this regard, the literature review was conducted with clearer methodology than an average Master's thesis in the field of Computer Science and Engineering at Aalto University.

There were two reasons for modifying the literature review to be less strict than in the study of Staples & Niazi. On one hand, doing a systematic literature review is very time-consuming and it would have increased the workload of the thesis too much to be feasible. On the other hand, using the systematic review guidelines as described by Staples & Niazi would have required knowing a lot about the topic beforehand to be able to finalize the research questions before doing the literature search. Using a slightly less strict way of conducting a literature review allowed me to learn new views about the topic and improve my research questions during the process of writing the thesis. I considered that the iterative approach of improving research questions and searching for more literature during the whole process provided me with a broader understanding of the topic than I would have gained by deciding all the research questions in the beginning of the process and not modifying them along the way. The downside of my approach is that it makes the literature review slightly more difficult to reproduce, but I consider that the upsides of my approach were more important than the downsides.

**Empirical research:** The empirical research only covered one project of one product development team in one corporate environment. Examining an effect in one unit, one setting or one context is a threat to external validity of research (Shadish, Cook, & Campbell, 2002).

Answering to the research questions required dealing with complex phenomena that were related to human behaviour and organizational characteristics. For researching such phenomena, social and educational scientists have developed qualitative research methods (e.g. interviewing) that try to tackle these challenges that are hard to research with quantitative methods (Seaman, 1999). It is also worth noting that due to the small size of the team whose methods and practices were being studied, it would have been difficult to get lots of data for making quantitative analysis.

Theories that interact with social psychological phenomena are difficult to reproduce since reproducing the exact conditions of the original study is difficult. However, raising the abstraction level of describing the phenomena with concepts makes the theory more generalizable. (Corbin & Strauss, 1990)

Action researchers usually have some prior knowledge (preunderstanding) of the topic of their research. While having prior knowledge can be beneficial (e.g. an action researcher may know which people have some useful information about a certain topic), having too much prior knowledge may lead an action researcher to assume too much about the researched phenomena. Additionally, having both the organizational role and researcher role may affect the relationships of the action researcher with the other members of the organization. (Coghlan, 2001) A typical problem that is faced with action research is that repeatability, reductionism and refutation are not very good (Baskerville, 1999).

Using semi-structured interviews may have made the challenge of assuming too much about the data due to being close to data less significant, because as described in chapter 3, semi-structured interviews are done by assuming very little about the knowledge of the interviewee and using the information that the interviewee supplies as a foundation for making further questions to reveal more information about the topics of the interview. (Wood, 1997) In contrast, fully structured interviews might have suffered more from researcher's assumptions, because in fully structured interviews all the questions are written beforehand and additional clarifying questions are not asked during the interview regardless of whatever information the interviewees tell during the interview.

In the interviews, feedback about the interviews and interview questions was asked at the end of the interview, which made it possible to improve the interview questions in case if an interviewee noticed a flaw in interview questions or if some useful information was not asked for in the interview.

In an ideal case, analysis of interview data should be done iteratively so that theory building happens iteratively during each interview so that the data that is gathered from a new interview is compared with the theory that is built based on the concepts that have been discussed in the earlier interviews (Corbin & Strauss, 1990). Unfortunately, the schedule for arranging interviews was so tight that there was not enough time for doing thorough analysis of the interview data between the interviews.

Doing both interviews and action research with observation was useful. Observation allowed the researcher to see how prototyping and agile requirements engineering were done in the project, whereas interviews provided a chance to ask project participants why these things were done in a certain way. In some cases, interviewees were not able to describe all the aspects of requirements engineering work and prototyping. Observation allowed the researcher to fill the gaps in the information that was gathered with interviews.

The empirical study examined the researched the topic retrospectively. In retrospective research, it is possible that some participants do not remember all the details of the experiments and some memories may have been distorted over time.

## 5.2 RQ1: How to improve prototyping for fast-paced agile software development?

Fast-paced agile software development poses many challenges for prototyping. Prototypes need to be easy to modify to support frequent requirement changes and fast to create so that they do not slow down the pace of the project. Additionally, agile projects usually have very little documentation and they rely heavily on informal communication, which can make it difficult to have a good overall picture of all the requirements that the prototypes are supposed to model. Prototyping can be improved for fast-paced agile software development in several ways, which are described below.

**Use large product-level or epic-level prototypes to make big picture clear:** Understanding the big picture was reported to be a problem in agile projects according to literature and empirical research. The results of empirical research indicate that large product-level or epic-level prototypes can help understanding the big picture up to the level of the product itself. However, product-level or epic-level prototypes do not help understanding how the product is aligned with company's strategy and vision. To solve this issue, it is possible to e.g. arrange workshops to review the strategy and vision with the team and create a simple prototype with high-level of abstraction that explains how the strategy and vision are related to the product.

**Use focused prototypes to iterate quickly:** Speed to create and modify prototypes are essential for efficient usage of prototyping in fast-paced agile projects. Small focused prototypes are faster to create and modify than large product-level or epic-level prototypes and therefore they can be useful for iterating a design of some specific interactions or user interface elements. It is worth noting that before starting to iterate a design as a focused prototype, it is important to consider what is the right focus of the prototype; spending time to iterate design of those parts of the user interface that are not along the most important user flows is usually just waste of time. However, iterating the design of the most important user flows (e.g. problematic parts of payment process in an online store) as focused prototypes makes sense and can save time when compared with creating a large prototype that contains the same features.

Because focused prototypes only contain some specific interactions and user interface elements, frequent changes of requirements in agile projects do not cause a need to modify the prototype after every requirement change; if the changed requirement is not in the focus of the prototype, it prototype does not need to be modified.

Software developers commented in interviews that they would prefer to receive design handoffs as component-level prototypes because software is usually modelled as components in code.

Empirical research participants commented that while focused prototypes are useful for product development organization's internal discussions (because people within the organization know the context of the prototype), they would be hesitant to use focused prototypes in usability tests because end users do not usually know the context of the focused prototype.

**Use appropriate prototype fidelity and simplified prototypes:** Low-fidelity prototypes are usually fast to create and therefore they can be useful for fast-paced agile projects. However, depending on the used tools, high-fidelity prototypes can be quite fast to modify and some parts of a high-fidelity prototype can often be used as basis for creating a new version of the prototype. When dynamic prototypes with personalized content are needed, coded high-fidelity prototypes can be the most viable option if it is necessary to make the content that is displayed in the user interface feel realistic e.g. for the purposes of usability testing.

Simplified prototypes and low-fidelity prototypes can be useful for keeping the expectations of development speed realistic; empirical research and literature suggest that polished high-fidelity prototypes can cause too high expectations of development speed. Additionally, simplified prototypes and low-fidelity prototypes do not suffer so much from frequent requirement changes in agile projects as detailed high-fidelity prototypes because if the prototype does not contain the information that needs to be modified after a requirement change, there is no need to spend time for modifying the prototype.

Efficient usage of prototyping in agile projects requires utilization of multiple prototyping approaches (large prototypes, small focused prototypes and different prototype fidelities); there is no one-size-fits-all approach that would be efficient for all the situations.

**Involve core team and relevant stakeholders in the prototyping process:** Agile projects do not usually have detailed written requirements specifications. Additionally, information about the requirements is often exchanged in form of oral communication. These aspects make it difficult for a single person (e.g. a user experience designer) to know all the requirements thoroughly. It was noticed in the action research that involving more people in the prototyping process might have resulted in a better end result. For example, the prototyping process could be started with a workshop, in which team members and relevant stakeholders sketch various ideas on paper and then create a paper prototype together. Then, a higher fidelity prototype could be created by a user experience designer or a front-end developer if necessary.

**Improved tooling is needed:** It was discovered in interviews and action research, that prototyping dynamic user-specific personal data is difficult and slow with typical visual prototyping tools. Better new tools would be needed for customizing the prototypes for different end users for the purposes of usability testing. As a workaround, it is possible to create coded prototypes that can be personalized more easily, but it would be useful to also make it easier to customize the prototypes for different users in visual tools, because it would make it easier for non-developers to create prototypes that display realistic content.

**Utilize parallel experimentation:** When the speed of the design process is critical (as it is in fast-paced agile projects), creating multiple prototypes at once and testing them can help evaluating multiple design ideas at a rapid pace. Additionally, using multiple prototypes in usability testing can help receiving more realistic feedback about the prototypes.

**Treat prototypes as disposable experiments and communicate it clearly:** Time pressure in fast-paced agile projects can lead to utilization of prototype implementations of product in production software, which may lead to costly quality issues later. Moreover, non-technical stakeholders may not understand that prototypes are supposed to be disposable experiments. To mitigate the issue, it is necessary to communicate the purpose of the prototypes clearly.

## 5.3 RQ2: How can prototyping support agile requirements engineering?

According to literature and empirical research, agile requirements engineering practitioners have to face several challenges. Prototyping can help with some of these challenges, but it is not a panacea that can solve all the problems of agile requirements engineering. For each main challenge, the effect of prototyping is described below. A summary of the findings of available in table *5.1*.

*Table 5.1:* Effect of prototyping on agile requirements engineering challenges based on literature and empirical research.

| Agile requirements engineering challenge | Effect of prototyping |
|---|---|
| Little documentation | Clear positive effect |
| Motivation issues to do constant requirements engineering work | Positive effect |
| Achieving quality communication with customer | Some positive effect |
| Missing big picture | Some positive effect on product-level big picture |
| Not enough emphasis on non-functional requirements | Small positive effect or clear negative effect, depends on how the prototype is created and what is the focus of the prototype |
| Challenges with prototyping itself | Clear negative effect, can be mitigated with good communication |

**Little documentation:** In agile projects, the amount of documentation is usually quite small, which may make it difficult for a customer to trust the product development team without explicit knowledge of the desired end result of the process beforehand. Moreover, it can be difficult to verify if the product satisfies the requirements. Because prototypes can act as user

interface documentation, prototypes can make it easier to understand the requirements when comprehensive written documentation does not exist.

**Motivation issues to do constant requirements engineering work:** Agile methodology embraces frequent change of requirements, which requires the team to do constant requirements engineering work. Keeping visual user interface documentation up-to-date is regarded as more motivating than updating written documentation according to literature an empirical research. Keeping the changed the requirements properly synchronized with tests and acceptance tests is challenging. In literature, it is suggested that acceptance tests can be based on prototypes.

**Achieving quality communication with customer:** Agile requirements engineering relies heavily on having good communication with customer. Prototypes can help visualizing the design of the product to make communication easier and to visualize the effect of suggested requirement changes that stakeholders propose. However, active customer presence is still needed to reach good communication and prototypes cannot solve all the potential communication challenges.

**Missing big picture:** Focusing on time-to-market prioritization and short-term planning can make it difficult to understand the big picture in agile projects. It was discovered in the empirical research that large product-level prototypes and epic-level prototypes can help the team understanding product-level big picture. However, these prototypes do not help the team understanding how the product is supposed to support company-level strategy and vision. Therefore, prototyping cannot solve the challenge of not having a clear big picture completely.

**Not enough emphasis on non-functional requirements:** Even though some interviewees commented that prototypes can help understanding some non-functional requirements at least in certain cases, prototyping does not solve the problem of ignoring or forgetting non-functional requirements completely. In fact, prototyping can make the issue worse if prototypes only focus on short-term planning of user-facing features.

**Challenges with prototyping itself:** Prototyping, a commonly used agile requirements engineering practice, can also cause some problems itself. Quick creation of polished high-fidelity prototypes may cause non-technical stakeholders to have unrealistic expectations of product development's speed. When creating prototypes in code, there is a temptation to ignore quality requirements and save development time by using prototype code in production software. To mitigate these issues, good communication is needed to explain that prototypes are supposed to be quick experiments and designing and developing the real product will take longer time than creating a prototype.

## 5.4 Model of agile requirements engineering with prototyping

As a result of analysis of empirical research and literature review, a model was created to describe current usage of prototyping with agile requirements engineering (section 5.4.1). Based on findings from literature and empirical research, a new improved model in presented in section 5.4.2.

### 5.4.1 Current model

The current model of requirements engineering within product development iterations (figure 5.1) consists of four main phases: elicitation, analysis, representation and validation. Prototyping is used for requirements representation and validation.

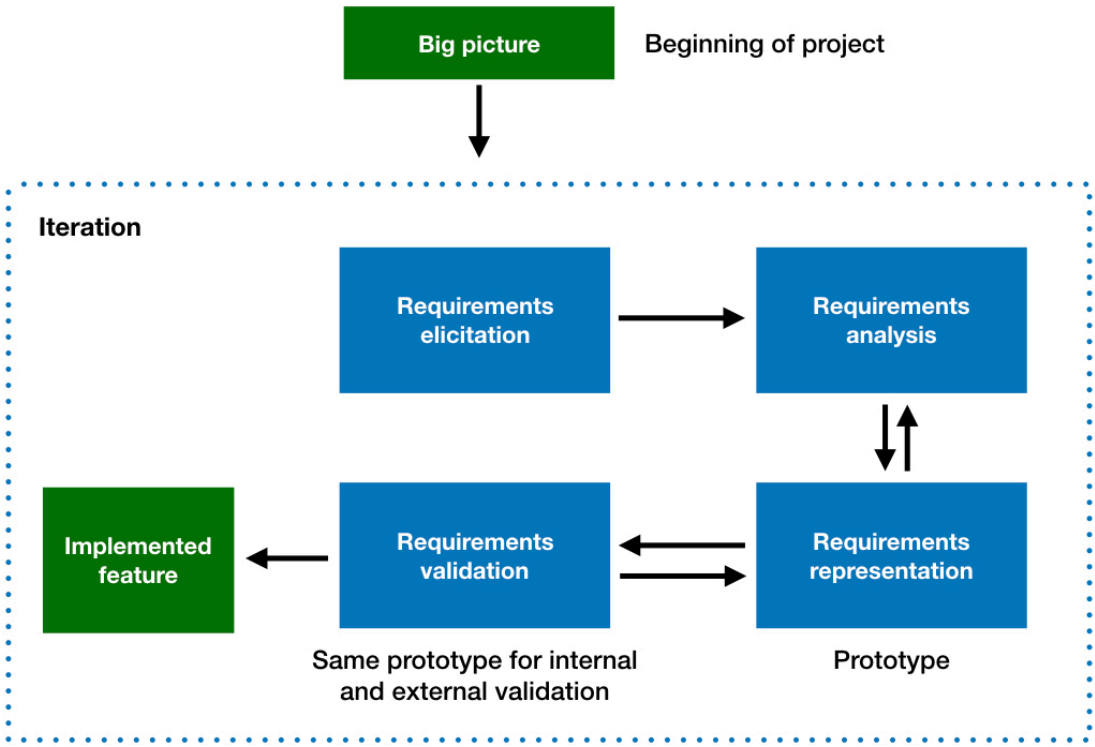


*Figure 5.1:* Current model of agile requirements engineering with prototyping.

**Requirements elicitation:** Ideas of new features (or modification of existing features) are usually discussed at a sprint planning session with the customer. These sessions are usually attended by the whole core team so that all the team members understand what the team is supposed to accomplish during the sprint. The features are typically listed on a backlog (i.e. a prioritized list of features), which is separated to a sprint backlog (a list of features for the current sprint) and a product backlog (a list of features for the upcoming sprints). Ideas for requirements can be based on product development organization members' ideas, customer feedback, analytics or some other source.

**Requirements analysis:** Analysis of requirements is usually done at the sprint planning session. To understand what implementing the requirement would require and to decide if the feature will be implemented, feasibility of the suggested requirements is analysed from multiple points of view: financial feasibility, technical feasibility and potential effects on user experience. Requirements are also prioritized.

**Requirements representation:** In its simplest form, a backlog item can be represented as e.g. a user story, a name of a code module or some other textual form. For requirements that involve user interface changes, it is usually necessary to model it in a visual form: in these cases, the requirement can be modelled as a prototype or a user interface composition (an image of how the user interface will look like). Depending on the need, the prototypes are usually clickable high-fidelity prototypes or low-fidelity paper prototypes. When prototypes are used, one prototype usually models a whole epic or product.

**Requirements validation:** First, the requirement is validated internally within the product development organization. The internal validation phase usually includes discussion with product owner, user experience specialists, software developers and potentially other relevant stakeholders.

After the internal validation, the requirement can be validated externally with end users. Typically, external validation happens in the form of usability testing or A/B testing.

If a problem is noticed with a prototype during the validation phase (e.g. it is noticed that the prototype has usability issues), the prototype is modified. The modifications are validated internally at first. If the modifications are large

enough, the prototype will go through a new round of external validation. Sometimes, if the estimated effort that is needed to modify a prototype is considered too large, the design is iterated in the form of user interface compositions instead of as a prototype.

After validation, the actual technical implementation of the requirement can be done. Design handoffs that explain user interface requirements to developers are usually done with a design tool plugin that can export user interface specifications so that software developers can inspect the attributes of each user interface element by simply clicking the elements on the screen. If it is necessary to document animations or transitions to developers, the design handoff may include a prototype, a video of the animations or transitions and written instructions that include timing properties of the animations or transitions.

**Understanding big picture:** In the beginning of a new product development project, the initial goals of the new product are defined and the initial feature set that supports the goal is planned. Due to this planning of iterations based on the goals of the product, the product development team has at least some understanding of the big picture. However, as time goes by and new features are added, the understanding of the big picture becomes more ambiguous when the team focuses on thinking of features that are needed in the near future.

### 5.4.2 New improved model

For the most part, the new improved model of agile requirements engineering with prototyping (figure 5.2) is the same as the current model. However, the new model contains the following differences:

**More versatile usage of different prototyping approaches:** When a problem is noticed with a prototype in requirement validation, the design of the problematic part of the user interface can be iterated quickly as a small focused prototype or a low-fidelity prototype. The designs of the focused prototypes and low-fidelity prototypes can be discussed internally within the product development organization (internal evaluators do not need to see all the details of the product because they know the context of the prototype). Then, when the iterated user interface pieces have been validated internally,

the modified designs can be brought back to the large prototype for the purposes of external validation.

For product development organization's internal discussions, it can be useful to use more low-fidelity prototypes and focused prototypes. Simplified prototypes and low-fidelity prototypes can be used to keep the expectations of non-technical stakeholders realistic.

Because the interviewed software developers indicated in the empirical research phase that they would like receive user interface specifications in the form of component-level prototypes, it might be useful to include small component-level prototypes in design handoffs to explain the user interface requirements clearly to developers.

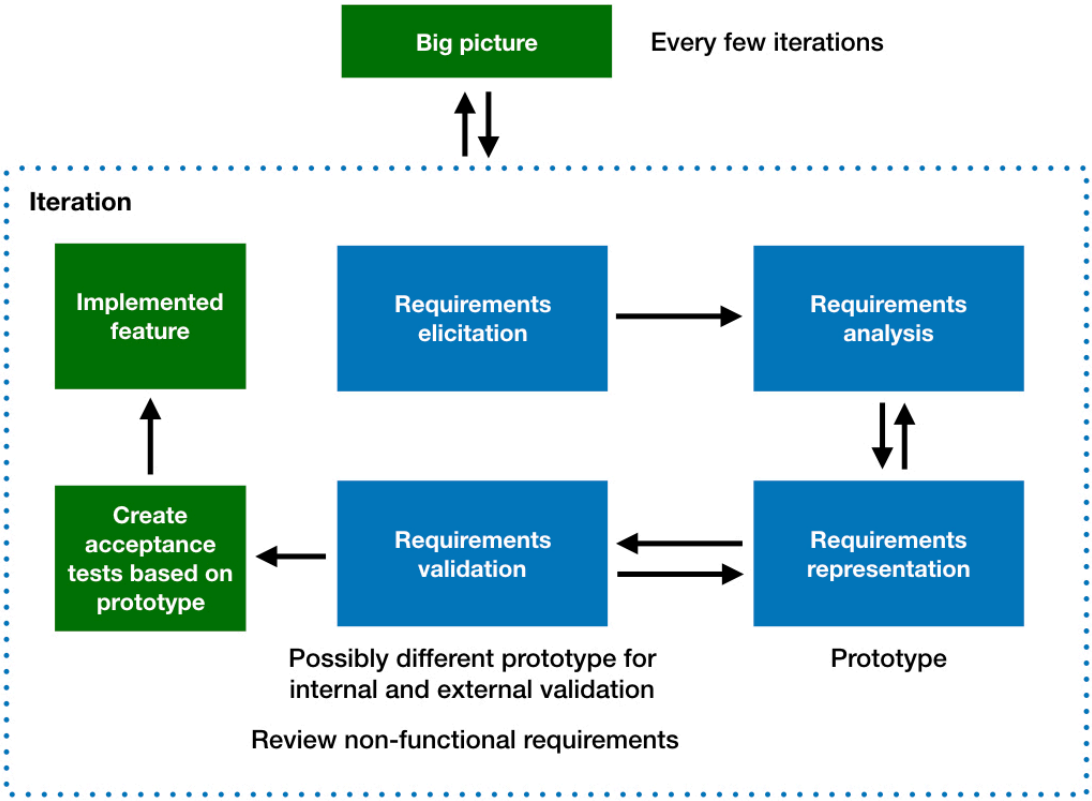


*Figure 5.2:* New model of agile requirements engineering with prototyping.

**Understanding big picture:** To keep the big picture clear during the whole project, it would be useful to arrange regular reviews of the big picture with the

core team. These reviews could be arranged in the form of workshops or design sprints, as it was suggested by the interview participants. At least one software developer should attend each of these sessions so that the technical team members would have better understanding of the high-level goals of the project. After the big picture has been reviewed, it would be useful to model it as a simplified prototype or some other similar form.

**Taking non-functional requirements into account more explicitly:** Because non-functional requirements are often forgotten in agile requirements engineering, it would be useful to review the non-functional requirements regularly. For example, each sprint planning session could include a quick review of the non-functional requirements of all the requirement changes that are suggested to be included in the sprint. Where applicable, prototypes could also be used in the review process of non-functional requirements.

**Linking user interface requirements and acceptance testing:** After user interfaces are modelled as prototypes or user interface compositions, it would be useful to create acceptance tests based on the prototypes or user interface compositions. This explicit linking of requirements and acceptance testing would ensure that features that have been acceptance tested correspond to the user interface requirements that have been modelled by a user experience designer.

# Chapter 6 Conclusions

## 6.1 Analysis of research problem

The research problem of this thesis was defined as follows:

> *How can different types of prototyping approaches support fast-paced product development in an agile software development project?*

The results of this study indicate that the chosen prototyping approach affects how well prototyping can be utilized in agile software development projects. There is no one-size-fits-all approach that is the most suitable approach for all the situations; a combination of multiple approaches is needed in order to utilize prototyping effectively in fast-paced agile projects.

Prototyping can help with various agile requirements engineering challenges. A prototype can act as documentation and it can help with communication challenges. Additionally, prototyping can have some effect on motivating product development team to constant requirements engineering work by making it clear, which requirements have changed and how they have changed. Large product-level or epic-level prototypes can also help with understanding product-level big picture. Prototyping can also have some effect on other agile requirements engineering challenges, such as neglected non-functional requirements, but the effect is not so prominent as with the previously mentioned challenges.

Unfortunately, prototyping can also be harmful from the point of view of requirements engineering. If prototypes focus on short-term user-facing features with heavy time-to-market prioritization, prototyping can have various negative effects: non-functional requirements may be forgotten and

the team may lose clear understanding of the big picture. Moreover, prototyping may cause unrealistic expectations to non-technical stakeholders and quality requirements may be ignored when using prototype code in production software in order to save development time.

Because the speed of prototype creation and modification is essential in fast-paced agile projects, it is important to consider, how different prototyping approaches affect the speed of creating and modifying the prototypes. Even though large product-level and epic-level prototypes have their benefits for understanding the big picture, they are not always a good solution for fast-paced projects; small focused prototypes can be faster to create and iterate. Additionally, large prototypes that contain lots of features may need to be modified often in agile projects, in which requirements change frequently. Focused prototypes that only contain some specific interactions or features do not need to be modified if the changed requirements do not belong to the focus of the prototype.

Low-fidelity prototypes and simplified prototypes share some of the same benefits that the focused prototypes have: they are usually fast to create and if they do not contain the details that are included in the changed requirements, they do not need to be modified in order to stay up-to-date with the latest requirements. However, low-fidelity prototypes and simplified prototypes have their limitations: sometimes, it is necessary to use high-fidelity prototypes and complex detailed prototypes. Additionally, when creating new versions of existing digital high-fidelity prototypes, some parts of the existing prototypes can usually be reused easily. Moreover, digital (high-fidelity) prototypes can usually be shared easily with stakeholders, whereas sharing a paper prototype with a large number of stakeholders can be cumbersome.

It was noticed in the empirical research phase that creating prototypes that display personalized user-specific content was quite slow with even the latest commercial prototyping tools. Improved tooling is needed to make prototyping of personalized software applications work better in fast-paced software projects.

## 6.2 Conclusions and recommendations

Prototyping can help with many agile requirements engineering challenges. However, it is not a panacea for all the challenges that are faced regularly in agile software development projects. It is a good idea to complement prototyping with other practices and methods to deal with those situations where prototyping is not helpful.

A new model for agile requirements engineering with prototyping is introduced in section 5.4.2. The new model suggests doing prototyping with more versatile approaches based on their suitability for each particular situation, reviewing big picture regularly and creating an explicit link between acceptance tests and prototypes.

The main idea of the model of improving prototyping for fast-paced agile projects is to use a dual model that includes large prototypes that help with understanding the big picture and small focused prototypes that make iterating the design of specific interactions or user interface elements fast so that prototyping does not slow down the velocity of fast-paced software projects. Moreover, this study recommends utilization of participatory design to involve the whole product development team (and relevant stakeholders) in the prototyping process.

The results of the empirical study are very similar to the ones that are described in literature. When comparing this study with the study of Käpyaho & Kauppinen (2015), the main difference is that according to this study, prototyping can support understanding product-level big picture if large product-level or epic-level prototypes are used. According to Bjarnason et al., (2011), not having an understanding of big picture can be a problem during the early phases of a project. In this study, it was noticed during the interviews that big picture was clearer during the early phases of the project because arranging design sprints at the beginning of the project could help understanding the big picture. This difference in the findings of this study and in the study of Bjarnason et al. may be caused by two reasons. First, Google Ventures 5-Day Design Sprint method that we use for our design sprints was introduced in 2016 (Knapp, Zeratsky, & Kowitz, 2016), five years after the study of Bjarnason et al. Secondly, it is also possible that Bjarnason et al., focused more on a technology-centered view of big picture, whereas the empirical phase of this study focused more on the user experience designers' point of view of understanding big picture (from technological point of view, not knowing all

the future requirements can be a difficult challenge in terms of system architecture, whereas from user experience designers' point of view the problem may not be so significant).

## 6.3 Future research needs

Because the empirical research phase of this thesis only covered one project, it is necessary to test generalizability of findings in more projects in the future.

One approach to prototyping that is described in literature by Schrage (1993) that may be useful for agile software projects (but which was not tested in the empirical phase of this study due to limited resources) is continuous prototyping. In more matured fields of design, such as automotive design and mobile phone hardware design, prototypes are produced periodically so that a recently created prototype, which matches the latest design, is always available. In fact, this approach to prototyping is very similar to usage of Continuous Integration (CI) and Continuous Deployment (CD) in the field of software development, which guarantee that a recent version of production-ready software is always available and can be deployed easily. Given the wide use of CI and CD in modern software projects, it is quite surprising that continuous prototyping has not already gained wide adoption in design of computer user interfaces.

# Bibliography


Agile Alliance. (2001). Agile Manifesto - Manifesto for Agile Software Development. *The Agile Manifesto*, 2001. https://doi.org/10.1177/004057368303900411

Baskerville, R. L. (1999). Investigating Information Systems with Action Research. *Communications of the Association for Information Systems*, *2*(3), 1–32. Retrieved from http://portal.acm.org/citation.cfm?id=374476

Bjarnason, E., Wnuk, K., & Regnell, B. (2011). A case study on benefits and side-effects of agile practices in large-scale requirements engineering. In *Proceedings of the 1st Workshop on Agile Requirements Engineering - AREW '11* (pp. 1–5). https://doi.org/10.1145/2068783.2068786

Boehm, B. (2000). Requirements that handle IKIWISI, COTS, and rapid change. *Computer*, *33*(7), 99–102. https://doi.org/10.1109/2.869384

Bogers, M., & Horst, W. (2014). Collaborative prototyping: Cross-fertilization of knowledge in prototype-driven problem solving. *Journal of Product Innovation Management*, *31*(4), 744–764. https://doi.org/10.1111/jpim.12121

Bolchini, D., Pulido, D., & Faiola, A. (2009). Paper in screen prototyping: an agile technique to anticipate the mobile experience. *Interactions*, *16*(4), 29–33. https://doi.org/10.1145/1551986.1551992

Brown, T. (2008). Design thinking. *Harvard Business Review*, *86*(6). https://doi.org/10.1145/2535915

Cockburn, A. (2002). Agile software development. *ELearning*, *5*(1), 97. https://doi.org/10.1109/2.947100

Coghlan, D. (2001). Insider Action Research Projects Implications for Practising Managers. *Management Learning*, *32*(1), 49–60. https://doi.org/10.1177/1350507601321004

Corbin, J., & Strauss, A. (1990). Grounded Theory Research: Procedures, Canons and Evaluative Criteria. *Zeitschrift Fuer Soziologie*, *19*(6), 418–427. https://doi.org/http://dx.doi.org/10.1007/BF00988593

Coyette, A., Kieffer, S., & Vanderdonckt, J. (2007). LNCS 4662 - Multi-fidelity Prototyping of User Interfaces. *LNCS*. https://doi.org/10.1007/978-3-540-74796-3_16

de Sá, M., Carriço, L., Duarte, L., & Reis, T. (2008). A mixed-fidelity prototyping tool for mobile devices. In *Proceedings of the working conference on Advanced visual interfaces - AVI '08* (p. 225). https://doi.org/10.1145/1385569.1385606

Drews, C. (2009). Unleashing the Full Potential of Design Thinking as a Business Method. *Design Management Review*, *20*(3), 38–44. https://doi.org/10.1111/j.1948-7169.2009.00020.x

Haugset, B., & Stålhane, T. (2012). Automated acceptance testing as an agile

requirements engineering practice. In *Proceedings of the Annual Hawaii International Conference on System Sciences* (pp. 5289–5298). https://doi.org/10.1109/HICSS.2012.127

Houde, S., & Hill, C. (1997). What do prototypes prototype? *Handbook of Human Computer Interaction*, 1–16. https://doi.org/10.1016/B978-044481862-1.50082-0

Käpyaho, M., & Kauppinen, M. (2015). Agile requirements engineering with prototyping: A case study. In *2015 IEEE 23rd International Requirements Engineering Conference, RE 2015 - Proceedings* (pp. 334–343). https://doi.org/10.1109/RE.2015.7320450

Knapp, J., Zeratsky, J., & Kowitz, B. (2016). *Sprint: How To Solve Big Problems and Test New Ideas in Just Five Days*. Simon & Schuster.

McCurdy, M., Connors, C., Pyrzak, G., Kanefsky, B., & Vera, A. (2006). Breaking the Fidelity Barrier - An Examination of our Current Characterization of Prototypes and an Example of a Mixed-Fidelity Success. *Proceedings of the International Conference on Human Factors in Computing Systems (CHI'06)*, 1233–1242. https://doi.org/10.1145/1124772.1124959

Paetsch, F., Eberlein, A., & Maurer, F. (2003). Requirements engineering and agile software development. *WET ICE 2003. Proceedings. Twelfth IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises, 2003.*, 308–313. https://doi.org/10.1109/ENABL.2003.1231428

Ramesh, B., Cao, L., & Baskerville, R. (2010). Agile requirements engineering practices and challenges: an empirical study. *Information Systems Journal*, *20*(5), 449–480. https://doi.org/10.1111/j.1365-2575.2007.00259.x

Ries, E. (2011). The Lean Startup. *Book*, 336. https://doi.org/23

Schrage, M. (1993). Culture(s) of Prototyping. *Design Management Journal (Former Series)*, *4*(1), 55–65. https://doi.org/10.1111/j.1948-7169.1993.tb00128.x

Seaman, C. B. (1999). Qualitative methods in empirical studies of software engineering. *IEEE Transactions on Software Engineering*, *25*(4), 557–572. https://doi.org/10.1109/32.799955

Shadish, W. R., Cook, T. D., & Campbell, D. T. (2002). Experimental and Quasi-Experimental Designs for Generalized Causal Inference. *Handbook of Industrial and Organizational Psychology*, *223*, 623. https://doi.org/10.1198/jasa.2005.s22

Snyder, C. (2003). *Paper Prototyping: The Fast and Easy Way to Design and Refine User Interfaces. Paper Prototyping: The Fast and Easy Way to Design and Refine User Interfaces*. https://doi.org/10.1016/B978-1-55860-870-2.X5023-2

Staples, M., & Niazi, M. (2007). Experiences using systematic review guidelines. *Journal of Systems and Software*, *80*(9), 1425–1437. https://doi.org/10.1016/j.jss.2006.09.046

Still, B., & Morris, J. (2010). The blank-page technique: Reinvigorating paper prototyping in usability testing. *IEEE Transactions on Professional Communication*, *53*(2), 144–157. https://doi.org/10.1109/TPC.2010.2046100

Thomke, S., von Hippel, E., & Franke, R. (1998). Modes of experimentation : an innovation process — and competitive — variable. *Research Policy*, *27*(3), 315–332. https://doi.org/10.1016/S0048-7333(98)00041-9

Tohidi, M., Buxton, W., Baecker, R., & Sellen, A. (2006). Getting the Right

Design and the Design Right. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (pp. 1243–1252). New York, NY, USA: ACM. https://doi.org/10.1145/1124772.1124960

Wood, L. E. (1997). Semi-structured interviewing for user-centered design. *Interactions*, *4*(2), 48–61. https://doi.org/10.1145/245129.245134

# Appendix A. Interview questions

This appendix contains interview questions that were used in the empirical part of the study. The interviews were held in Finnish because all the interviewees spoke Finnish as their native language.

# Interview questions

***Interview duration**: 44–72 minutes (average duration: 59.6 minutes)*
***Interview type:** semi-structured interview*
***Subjects:** 4 software developers, 1 user experience design specialist*

***Introduction:***

- *Tell the themes (prototyping and agile requirements engineering) of the interview to the person who is being interviewed.*
- *Encourage interviewees to tell their own understanding and interpretations about the topic. Tell that there are no right or wrong answers to the questions.*
- *Ask permission to record the interviews with a voice recording app. Ask interviewees to sign letter of informed consent (appendix B). Inform interviewees about the purpose of the interviews and how the interview notes and recordings are stored. Tell that the names of interviewees are not shown in the Master's thesis.*

***Interview questions:***

- *Questions 1–3 gather background information and for "warm-up".*
- *Questions 4–12 focus on prototyping.*
- *Questions 13–21 focus on agile software development and agile requirements engineering.*
- *Questions 12, 20 and 21 aim at finding answers to research questions. Other questions serve as introduction to the topic and support conversation about the research problem and research questions.*
- *Questions 22–23 are targeted at improving interview questions iteratively during the empirical research phase and for asking feedback about the interview from the person that is being interviewed.*

1. Taustatiedot
    a. Ikä
    b. Sukupuoli
    c. Työtehtävä / rooli
    d. Työkokemuksen määrä ohjelmistoalalla
2. Kertoisitko lyhyesti, millaisia tehtäviä työhösi sisältyy?
3. Kertoisitko hieman tyypillisestä työpäivästäsi ja viimeisimmästä projektistasi?
4. Millaisissa tilanteissa olet käyttänyt prototyyppejä?
    a. Oletko joskus ollut itse mukana prototyypin tekemisessä?
5. Miten määrittelisit käsitteen *prototypointi* omin sanoin?

6. Mitä hyötyjä prototyyppien käytöstä on mielestäsi ollut tässä projektissa?
7. Onko prototyyppien käyttämisestä aiheutunut jotain negatiivisia puolia?
8. Onko prototyypeistä ollut mielestäsi hyötyä ohjelmistoprojektin vaatimusten ymmärtämisessä?
9. Prototyyppejä voidaan tehdä monella tavalla, esimerkiksi klikkailtavina prototyyppeinä tai paperiprototyyppeinä. Loppukeväällä käytimme klikkailtavia prototyyppejä ja paperiprototyyppiä käytettävyystesteissä. Mitä mieltä olet näiden kahden erilaisen prototypointitavan hyvistä ja huonoista puolista?
10. Oliko jompikumpi prototyyppi parempi käyttöliittymää koskevien vaatimusten ymmärtämiseksi? Millä tavoin? Miksi?
11. Millä tavoin prototypointia voitaisiin mielestäsi parantaa siihen nähden, miten sitä on käytetty tässä projektissa?
12. Projektimme on ollut ajoittain varsin nopeatempoinen. *(RQ1):* **Olisiko prototypointia mielestäsi mahdollista kehittää siten, että se tukisi paremmin nopeatempoista ohjelmistokehitystä?**
13. Onko *ketterä ohjelmistokehitys* käsitteenä tuttu?
14. Miten määrittelisit *ketterän ohjelmistokehityksen* omin sanoin?
15. Mitä eroa ketterällä ohjelmistokehityksellä on perinteiseen ohjelmistokehitykseen verrattuna?
    a. Oletko osallistunut molemman tyylisiin projekteihin?
16. Mitä hyviä ja huonoja puolia ketterässä ohjelmistokehityksessä on mielestäsi?
17. Oletko huomannut jotain haasteita ketterien ohjelmistoprojektien vaatimusten määrittelyssä?
    a. Mitkä ovat olleet mielestäsi suurimpia haasteita tässä projektissa? Millaiset asiat voisivat auttaa näiden haasteiden ratkaisemisessa?
18. Kertoisitko hieman siitä, miten ohjelmistoprojektin vaatimuksia on määritelty tässä projektissa?
    a. Miten vaatimusten muutokset ja tarkentaminen tapahtuvat?
19. Millä tavoin voisimme mielestäsi parantaa ohjelmiston vaatimusten määrittelyä?
20. *(RQ2):* **Onko prototyypeistä ollut hyötyä projektin vaatimusten määrittelyssä? Entä onko siitä ollut haittaa? Millä tavoin? Miksi?**
21. Keskustellaan muutamasta aiheesta, joiden on huomattu olevan haasteellisia ketterien ohjelmistoprojektien vaatimusten määrittelyn kannalta. Onko prototypoinnilla vaikutusta seuraaviin asioihin liittyen? Jos on, onko vaikutus positiivinen vai negatiivinen vai neutraali (ei selkeästi positiivinen tai negatiivinen)?
    a. Vähäisestä dokumentaatiosta johtuvat projektinhallinnan haasteet, esim. asiakkaan vaikeudet luottaa projektiin, josta ei

ole olemassa selkeää ”speksiä” tai vaikeus tehdä ohjelmiston hyväksymistestausta *(acceptance testing),* kun ohjelmiston vaatimuksia ei ole kirjattu selkeästi ylös.

b. Tiimin motivointi jatkuvaan vaatimusmäärittelytyöhön ja siihen, että vaatimukset muuttuvat usein.
c. Riittävän laadukas kommunikaatio asiakkaan kanssa (koska ketterät projektit korostavat suullista kommunikaatiota muodollisten vaatimusmäärittelydokumenttien laatimisen sijaan, kommunikaatio on tärkeää projektin onnistumisen kannalta).
d. Ei selkeää ymmärrystä isosta kuvasta.
e. Ei-toiminnallisten vaatimusten ja laatuvaatimusten määrittelyn unohtuminen tai liian vähäinen painoarvo, mikä voi vaikuttaa esim. ohjelmiston arkkitehtuuriin tai suorituskykyyn.
f. Prototypointiin liittyvät haasteet:
    i. Ei-teknisen asiakkaan liian suuret odotukset ohjelmistolle hienoista lopullisen näköisistä prototyypeistä johtuen.
    ii. Prototyyppiä varten luodun koodin käyttö lopullisessa tuotanto-ohjelmistossa.

22. Tuleeko mieleen jotain sellaisia tähän aiheeseen liittyviä kysymyksiä, joista minun olisi ollut hyödyllistä kysyä, mutta en kysynyt?
23. Avoin palaute haastatteluun liittyen.
    Jos sinulla on jotain kysyttävää haastatteluun liittyen, voit kysyä nyt vapaasti haluamistasi asioista tai esittää kommentteja haastattelun teemoihin liittyen.

# Appendix B. Letter of informed consent

Interview participants were asked to sign the letter of informed consent. A copy of the letter of informed consent is available on the following page. The letter is in Finnish because all the interviewees spoke Finnish as their native language.

**SUOSTUMUS TUTKIMUKSEEN OSALLISTUMISESTA**

**DIPLOMITYÖ**

Toni Karttunen, tekn. kand.
Master's Programme in Computer, Communication and Information Sciences
Aalto-yliopisto, Perustieteiden korkeakoulu
toni.karttunen@aalto.fi

Opinnäytetyön valvoja:
Professori Marko Nieminen
Aalto-yliopisto, Perustieteiden korkeakoulu
marko.nieminen@aalto.fi

Allekirjoittamalla tämän dokumentin annan luvan haastatella minua Aalto-yliopiston tietotekniikan opiskelija Toni Karttusen diplomityötä varten. Haastateltavien henkilöiden nimiä ja haastatteluissa mahdollisesti mainittujen henkilöiden nimiä ei mainita diplomityössä, ellei siihen pyydetä erillistä lupaa. Haastatteluista tehdyt äänitallenteet ja kirjalliset muistiinpanot arkistoidaan diplomityön tekemistä varten. Tiedot käsitellään luottamuksellisesti.

Paikkakunta: _____
Päivämäärä: ___ / ___ 2017

| | |
|---|---|
| _____ | _____ |
| Haastateltavan allekirjoitus | Diplomityön tekijän allekirjoitus |
| _____ | _____ |
| Nimen selvennys | Nimen selvennys |